

---

# **Intro to Docker Labs - F5 ISCFY17**

***Release 0.1***

**Nicolas Menant**

**Jul 08, 2020**



<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Topology</b>	<b>5</b>
<b>3</b>	<b>Connecting to UDF</b>	<b>7</b>
3.1	Start your environment . . . . .	7
3.2	Access your environment . . . . .	7
<b>4</b>	<b>Setting up</b>	<b>9</b>
<b>5</b>	<b>Lab 1: Run a container</b>	<b>11</b>
5.1	The legacy Application . . . . .	11
5.2	Docker Pull . . . . .	12
5.3	Docker Run . . . . .	13
5.4	Docker ps / inspect . . . . .	14
5.5	Docker stop . . . . .	15
<b>6</b>	<b>Lab 2: Building a container</b>	<b>17</b>
6.1	Setup . . . . .	17
6.2	Docker build . . . . .	19
6.3	Bonus Activity . . . . .	20
<b>7</b>	<b>Lab 3: Publishing a container</b>	<b>21</b>
7.1	Docker registry . . . . .	21
7.2	Docker tag . . . . .	21
<b>8</b>	<b>Lab 4: Docker Networking</b>	<b>23</b>
8.1	Ephemeral ports . . . . .	23
8.2	Linux Bridge Network . . . . .	24
8.3	Docker and networking . . . . .	26
<b>9</b>	<b>Appendix</b>	<b>29</b>
9.1	Changing display for HiRes displays . . . . .	29
<b>10</b>	<b>Indices and tables</b>	<b>31</b>



This document is to help you learn more about Docker. Though this session we will help you: \* Access a docker environment (hosted in UDF) \* Manipulate docker containers

- Run a container (Create, Start, Stop, Delete, Status)
- Create a container
- Publish a container
- Overview of Docker networking

Contents:





# CHAPTER 1


## Introduction

the following labs are a basic introduction to Docker containers and networking.


To do the labs, we will leverage UDF and the blueprint called ‘Introduction to Docker’.

**F5 CSI Marathon/Mesos**  
 Eric Chen  
🕒 a day ago 🚀 3  

 **\$2.42**  
Solution Per hour

 **DEPLOY** **DETAILS**

if you would prefer to have a pdf version of this guide, you may click on **v:latest** at the bottom of the column menu and download a pdf version of this guide

 Read the Docs **v: latest** ▼  
Versions  
**latest**  
Downloads  
PDF HTML Epub  
On Read the Docs  
Project Home Builds Downloads  
On GitHub  
View Edit  
Search





## CHAPTER 2

---

### Topology

---

The topology for this lab is more advanced than needed because it is leveraged also by another lab.

Here is the list of the blueprint components that we will use in this lab - everything else can be safely ignored:

- server01 node (ubuntu hosting docker)
- server02 node (ubuntu hosting docker)
- win2012 (jumpbox to the environment)



## CHAPTER 3

---

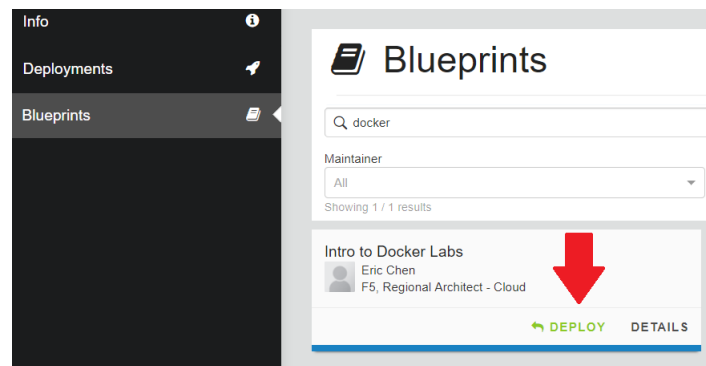
### Connecting to UDF

---

We consider that you have access to UDF for the different labs

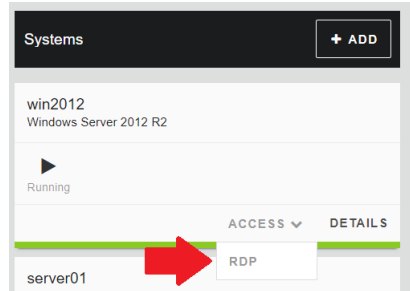
#### 3.1 Start your environment

If you are running this on your own, find the ‘Introduction to Docker’ blueprint and deploy it.



#### 3.2 Access your environment

Once your environment is started, find the ‘win2012’ component under ‘Components’ and launch RDP (in the ACCESS menu)



Click on the shortcut that got downloaded and it should open your RDP session. The credentials to use are under the Details tab.

**Warning:** For MAC user, it is recommended to use Microsoft Remote Desktop. You may not be able to access your jumpbox otherwise. It is available in the App store (FREE).

## CHAPTER 4

---

### Setting up

---

There are GUI interfaces to manage a Docker container, but to start we are going to learn about using the command-line interface. Most users will prefer to use command-line tools.

Open a session on server01. Double click on this icon on your desktop



You should see a large terminal window. If the Text is too small; please visit the Appendix section at the end of this document on changing the size of the text.

You should be automatically logged in. If not, use the following credentials:

- Login: ubuntu
- SSH Key: [Located on Desktop]



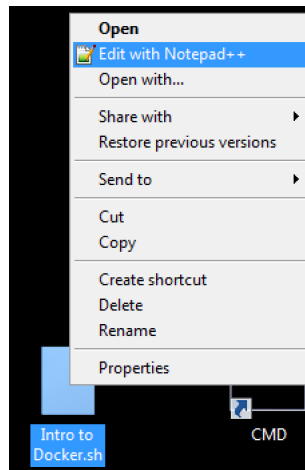
# CHAPTER 5

---

## Lab 1: Run a container

---

All commands from this lab will also be provided as a text file. You may want to download this file first and copy-and-paste the following commands. the commands are in a file on your desktop called *Intro to Docker.sh*. You may edit it by doing a right click on edit and select *Edit with Notepad++*



The first lab is an example of migrating an application to run in a container.

Goals of the lab

1. Learn how to launch an existing container
2. Create your own container
3. Distribute your container

## 5.1 The legacy Application

One of the use-cases for utilizing containers is to migrate an existing on-premises application to first run in a container and eventually migrate the container to run in a public cloud environment. The first lab will migrate the existing app

that is running at: <http://10.1.10.11/>

Launch Chrome and visit **<http://10.1.10.11>** (make sure to enter *http://*)



Created By: Your Name

Stats

PHP VERSION	HOSTNAME	SERVER IP:PORT	CLIENT IP
5.5.9-1ubuntu4.20	mesos-agent01.f5demo.com	10.1.20.101:80	10.1.20.4

10.1.10.11 is one of server01 IP addresses. You can check this by typing **ifconfig eth1** in your putty session

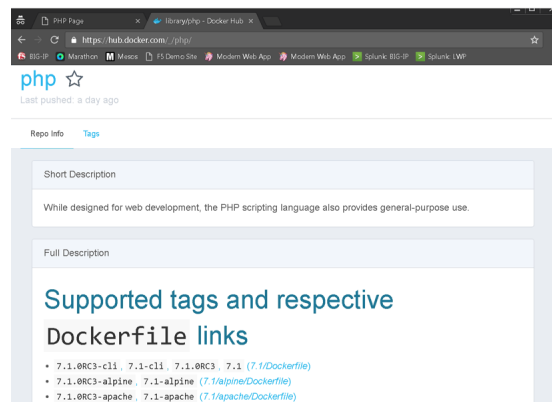
```
user@mesos-agent01:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.1.20.101  Bcast:10.1.20.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe01:9e00:12:34:56  Prefixlen 64
          UP BROADCAST RUNNING MULTICAST  MTU:1500
          RX packets:5083768 errors:0 dropped:0 overruns:0
          TX packets:4161069 errors:0 dropped:0 overruns:0
          collisions:0 txqueuelen:1000
          RX bytes:1900673296 (1.9 GB)  TX bytes:1900673296 (1.9 GB)
```

The first step of this lab is to download a container that has PHP.

## 5.2 Docker Pull

Normally you would run the command **docker pull [image]:[tag]** to pull down a public image of a container. This is similar to going to the F5 Downloads site to grab the latest vLab or ISO, but it doesn't require any authentication. This is an example of a low-friction method of obtaining software that is appealing to Mode 2 users.

The *Docker Hub* (transition to *Store*) has a listing of community images that are available. Visit **<https://hub.docker.com/>** and search for *php*.



You'll see a long list of available versions of PHP that can be downloaded.

In this lab, we have already downloaded the required containers. You can view the available containers by running the command:

```
docker images
```



```

user@mesos-agent01:~$ docker images
REPOSITORY          TAG                 IMAGE ID
php                  7-apache           f41e38c51e7c
php                  5.6-apache         8f9b7e57129a
registry             2                  541a6732eadb
registry             latest             541a6732eadb
centos               latest             980e0e4c79ec
registry:5000/f5demo <none>             f1717ca999cb
registry:5000/lwp-controller v0.1.1           60b6f67c97fd
php                  <none>             a84d6469e522
busybox              latest             2b8fd9751c4c

```

For the lab we have retrieved *php:5.6-apache* and *php:7-apache*. These represent containers that can run PHP 5.6 / 7 running on the Apache web server (httpd).

**Warning:** For your information, if the user doesn't have the proper privileges, you'll see something like this:

*Cannot connect to the Docker daemon. Is the docker daemon running on this host?*

In case of this error, you can either run as root or use the sudo command, i.e. **sudo docker images**. In this lab, it should not be the case. We have added the user *user* to the docker unix group to enable it to be able to run these commands as a non-root user.

## 5.3 Docker Run

The community PHP container by default does not have any content. You can verify this by running:

```
docker run -p 8080:80 --name myphp php:5.6-apache
```

This command will start the PHP 5.6 container. Some of the options we specified:

- The '-p 8080:80' indicates that we want to create a port forwarding rule to map the host port '8080' to the container port 80 (more about container networking later in the lab).
- The '--name myphp' is used to name the container. This is not required, but will make future steps in the lab easier.

**Warning:** You will see error messages like *Could not reliably determine the server's fully qualified domain name,...* this is expected.

You will see that you don't get a prompt back. This is expected. We just launched the container in foreground.

```

user@mesos-agent01:~$ docker run -p 8080:80 --name myphp php:5.6-apache
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.4.
ally to suppress this message
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.4.
ally to suppress this message
[Thu Oct 27 22:33:57.872799 2016] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.10 (Debian) PHP/5.6.26
ions
[Thu Oct 27 22:33:57.876010 2016] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'

```

Visit <http://10.1.10.11:8080> in Chrome and you will see the following error page (expected).



## 5.4 Docker ps / inspect

Now that you have a container running you may want to learn some additional docker commands.

Open a new terminal window on agent01 (**leave the existing window open**).



run the following command:

```
docker ps
```

You should see the following:

```
user@mesos-agent01:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
3b2f90f1e6bf       php:5.6-apache     "apache2-foreground" 2 minutes ago
bda984d1525d       registry:2         "/entrypoint.sh /etc/" 5 days ago
```

Note there are two containers that are running. The PHP container that you launched and a *registry* container that will be used later in this lab.

Note the 'Container ID' and 'Ports' columns. The 'Container ID' represents a unique identifier that you can use to manage individual containers and the 'Ports' columns lists what the current port forwarding mappings are:

Highlight the 'Container ID' for the PHP container (this will place the value into your copy and paste buffer - if you double click on the ID, Putty will automatically highlight it).

```
user@mesos-agent01:~$ docker ps
CONTAINER ID        IMAGE
3b2f90f1e6bf       php:5.6-apache
bda984d1525d       registry:2
```

Use this to run the command:

```
docker inspect [CONTAINER ID]
```

or you can simply run:

```
docker inspect myphp
```

This provides a large amount of detailed data about a container that can be useful if you need to troubleshoot any problems.

```

user@mesos-agent01:~$ docker inspect 3b2f90f1e6bf
[
  {
    "Id": "3b2f90f1e6bf1a12096fba5cfa05c27ffbc652daf31",
    "Created": "2016-10-19T08:23:43.023540879Z",
    "Path": "apache2-foreground",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 2623,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2016-10-19T08:23:58.557720637Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    }
  }
]

```

Now run:

```
docker logs myphp
```

This will output the logs from the container (this should match what you see in the other terminal open where we started this container).

## 5.5 Docker stop

There are two ways that you can stop the container that we started earlier. Either type **CTRL+C** to terminate the running container (from the window that you originally started it).

```

user@mesos-agent01:~/containers$ docker run -p 8080:80 --name myphp php:5.6-apache
docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.
See 'docker run --help'.
user@mesos-agent01:~/containers$ sudo docker run -p 8080:80 --name myphp php:5.6-apache
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.3
Abally to suppress this message
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.3
Abally to suppress this message
[Tue Oct 18 16:03:00.004606 2016] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.10 (Debian) PHP/5.6.25
started
[Tue Oct 18 16:03:00.006170 2016] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'

^C[Tue Oct 18 16:13:06.220107 2016] [mpm_prefork:notice] [pid 1] AH00169: caught SIGTERM, shutting down
user@mesos-agent01:~/containers$

```

Note that the web server logs are output to the screen (vs. a log file).

You could also do:

```
docker stop myphp
```

```

user@mesos-agent01:~$ docker stop myphp
myphp
user@mesos-agent01:~$

```

If you run this command, you will see that we got our prompt back in the other terminal session since we stopped this process.

If you run:

```
docker ps
```

you will no longer see *myphp* running.

Run:

```
docker ps
docker ps -a
docker rm myphp
docker ps -a
```

**Docker ps** only shows running containers. Adding **-a** will show stopped containers and **rm** will remove a stopped container.

```
user@mesos-agent01:~$ docker ps
CONTAINER ID        IMAGE               COMMAND
bda984d1525d        registry:2         "/entrypoint.sh /etc/"

user@mesos-agent01:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
3b2f90f1e6bf        php:5.6-apache     "apache2-foreground"
bda984d1525d        registry:2         "/entrypoint.sh /etc/"

user@mesos-agent01:~$ docker rm myphp
myphp
user@mesos-agent01:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
bda984d1525d        registry:2         "/entrypoint.sh /etc/"
user@mesos-agent01:~$
```

---

## Lab 2: Building a container

---

### 6.1 Setup

Now that we've covered the basics of running a container it is time to take a look at building our own custom container. For this lab we will use WinSCP to transfer files from the Windows client to the Linux host running Docker.

Launch the 'WINSCP' shortcut that is on the Desktop. Be sure to use this link, it should connect and place you in the Folder 'mycontainer'.



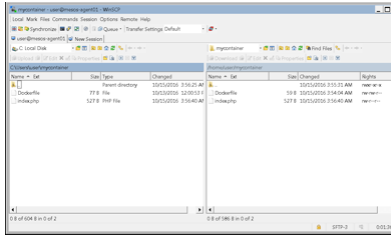
---

**Note:** If connection/authentication fails for some reason, here are the relevant information to launch your WinScp session:

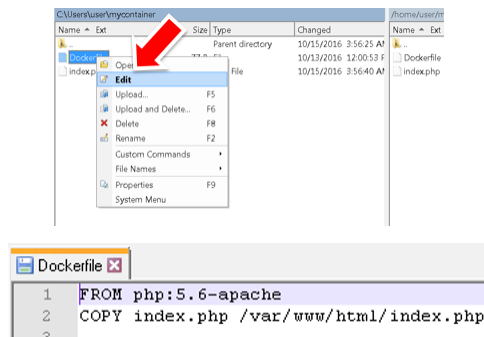
- hostname : 10.1.10.11
- login: ubuntu
- ssh key: [On the Desktop]

once logged in:

- on server01: go to /home/ubuntu/f5-intro-to-docker/mycontainer directory
  - locally: select your c:\Users\Administrator\Desktop\f5-intro-to-docker\mycontainer directory
-

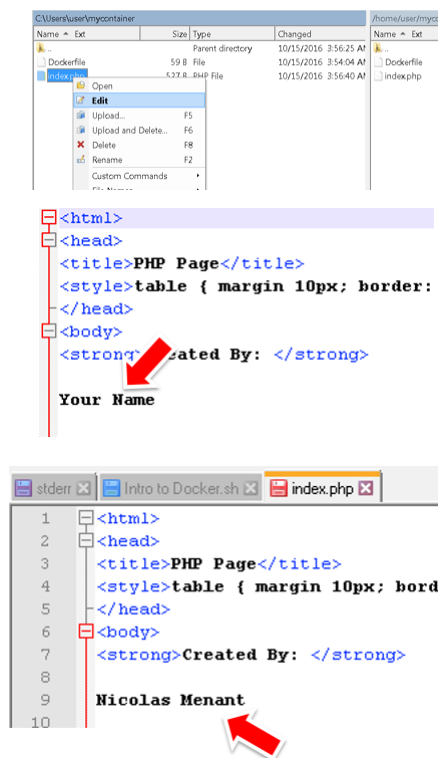


On the left panel, first open *Dockerfile* by right-clicking on the filename and selecting *Edit*.

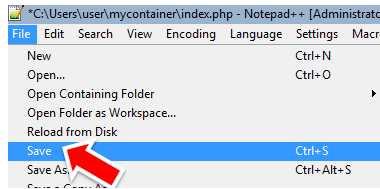


You should see a very simple Dockerfile. This file is used build a container. The first line references which container we want to use as the starting container and the second line references the file that we want to copy into the new container.

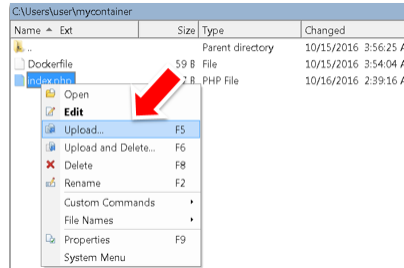
On the left panel, open 'index.php' and change the value of *Your Name* and click on the 'Save' button.



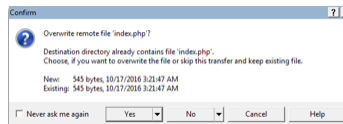
Save your changes and close Notepad++



Now upload your updated 'index.php'. We don't need to upload the Dockerfile file because we didn't change anything.



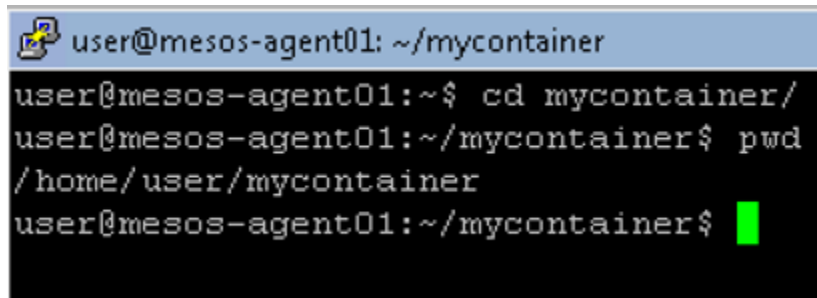
When prompted, click 'Yes' to overwrite the existing file.



## 6.2 Docker build

Back to your terminal window (on **server01**), run the following commands and verify that you're in the correct directory:

```
cd mycontainer
pwd
```



```
docker build -t mycontainer:5.6 .
```

**Note:** Note the '.' at the end of the command.


```
user@mesos-agent01:~/mycontainer$ docker build -t mycontainer:5.6 .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM php:5.6-apache
--> 8f9b7e57129a
Step 2 : COPY index.php /var/www/html/index.php
--> 7144089e3916
Removing intermediate container d56516a4cf86
Successfully built 7144089e3916
user@mesos-agent01:~/mycontainer$
```

This command specifies that you want to build a new container with the name *mycontainer* and the tag '5.6'.

Running **docker images** you should see your new container.

```
docker images
```

```
user@mesos-agent01:~/mycontainer$ docker images
REPOSITORY          TAG
mycontainer          5.6
php                  7-apache
php                  5.6-apache
registry            2
registry            latest
centos              latest
registry:5000/f5demo <none>
registry:5000/lwp-controller v0.1.1
php                  <none>
busybox             latest
user@mesos-agent01:~/mycontainer$
```




Verifies that it works by running

```
docker run -d --name myapp -p 8080:80 mycontainer:5.6
```

**Note:** The option '-d' makes the container run in the background. We get our prompt back.

We can check it is up and running by connecting to **http://10.1.10.11:8080** in Chrome.

Created By: Nicolas Menant			
<b>Stats</b>			
PHP VERSION	HOSTNAME	SERVER IP:PORT	CLIENT IP
5.6.26	15221af326c5	172.17.0.3:8080	10.1.20.4



**Note:** Pay attention to the difference in Server IP (server01 is 10.1.10.11)

You are now running a supported version of PHP on the same host that was previously running an unsupported version. Similar to the virtue of running vCMP; containers make it easier to run multiple versions of software on the same platform.

## 6.3 Bonus Activity

Rebuild mycontainer to run using the php:7-apache image. PHP 5.6 is also approaching end-of-life and PHP 7 is the most recent version! Do not delete the image mycontainer:5.6



---

## Lab 3: Publishing a container

---

### 7.1 Docker registry

The container is now running on mesos-agent01, but what if we want to have it run on server02? It is possible to manually export/import the image from one host to another, but it is more practical to use a Docker registry.

A Docker registry is an image repository of Docker containers. You can ‘push’ a container into the public Docker Hub or maintain your own private Docker registry/repository. For the lab we have previously created a registry that lives at ‘registry:5000’ (running on mesos-agent01).

---

**Note:** We already setup docker to use this registry in the /etc/default/docker file (need to be root to access it)

---

### 7.2 Docker tag

Currently ‘mycontainer:5.6’ lives locally on server01. We need to apply a tag that will indicate where we want it to go, then we need to push/copy the image to that location.

Run

```
docker tag mycontainer:5.6 registry:5000/mycontainer:5.6
docker images
```

---

**Note:** Note that you have two tags with the same Image ID.

---

```
user@mesos-agent01:~/mycontainer$ docker images
REPOSITORY          TAG          IMAGE ID
mycontainer          5.6          5ea5a0d1fc5b
registry:5000/mycontainer 5.6          5ea5a0d1fc5b
```

Now run:

```
docker push registry:5000/mycontainer:5.6
```

Open a terminal window to server02. You have the following shortcut on your desktop, use it.



Run

```
docker run --rm -p 8080:80 --name myapp mycontainer:5.6
```

```
user@mesos-agent02:~$ docker run --rm -p 8080:80 --name myapp mycontainer:5.6
Unable to find image 'mycontainer:5.6' locally
Pulling repository docker.io/library/mycontainer
docker: Error: image library/mycontainer:5.6 not found.
See 'docker run --help'.
```

Note that the command failed. The container does not exist on this host. Now run.

```
docker run --rm -p 8080:80 --name myapp registry:5000/mycontainer:5.6
```

---

**Note:** The option ‘--rm’ specify that the container should be automatically removed with it exits

---

```
user@mesos-agent02:~$ docker run --rm -p 8080:80 --name myapp registry:5000/mycontainer:5.6
Unable to find image 'registry:5000/mycontainer:5.6' locally
5.6: Pulling from mycontainer
```

The container was found on the private registry and was started. Verify by visiting <http://10.1.10.12:8080> in Chrome.

Created By: Eric Chen

**Stats**

PHP VERSION	HOSTNAME	SERVER IP:PORT	CLIENT IP
5.6.26	657ce455256e	172.17.0.2:8080	10.1.20.4

We are done with this container so we can delete it. Since we specified the ‘rm’ option, you just need to terminate the process. You can do so by doing

```
Ctrl+C
```

Make sure that it got removed with this command

```
docker ps -a
```

You can now close the agent02 terminal window. It will not be used for the rest of the lab.

Lab 4: Docker Networking

---

## 8.1 Ephemeral ports

Up to this point we have been using a static port mapping of port 8080 on the host to port 80 on a container. This works OK for a limited use-case, but generally you should not expect a container's port binding to be static.

Connect to **server01** via putty (use shortcut on Desktop) and run:

```
docker run -d --name myapp2 -p :80 mycontainer:5.6
docker port myapp2
```

```
user@mesos-agent01:~/mycontainer$ docker port myapp2
80/tcp -> 0.0.0.0:32771
```

---

**Note:** the port option allows you to see the port mappings that was done with the container.

---

Record the port value that is returned (your output will differ) and open a new Chrome window for **http://10.1.10.11:[PORT VALUE]**



The screenshot shows a web browser window titled 'PHP Page'. The address bar displays 'mesos-agent01:32771'. The page content includes a header 'Created By: Nicolas Menant' and a section titled 'Stats' containing a table with the following data:

PHP VERSION	HOSTNAME	SERVER IP:PORT	CLIENT IP
5.6.26	bb48a55fe729	172.17.0.4:32771	10.1.20.4

Now run:

```
docker restart myapp2
docker port myapp2
```

Observe that the port value has changed!

```
user@mesos-agent01:~/mycontainer$ docker restart myapp2
myapp2
user@mesos-agent01:~/mycontainer$ docker port myapp2
80/tcp -> 0.0.0.0:32772
```

## 8.2 Linux Bridge Network

From the previous labs you may have noticed that on both server01 and server02 the container is running in the 172.17.0.0/16 network. By default Docker will create a Linux Bridge network on the host called *docker0*.

Connect via putty to **mesos-agent01** and run:

```
ifconfig docker0
```

```
user@mesos-agent01:~/mycontainer$ ifconfig docker0
docker0    Link encap:Ethernet  HWaddr 02:42:c1:d9:35:0b
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:c1ff:fed9:350b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:378704 errors:0 dropped:0 overruns:0 frame:0
          TX packets:467626 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1085333701 (1.0 GB)  TX bytes:2231056351 (2.2 GB)
```

From Chrome visit **http://10.1.10.11:[PORT VALUE]** (port value from last lab step) and record what the Server IP value is.



Created By: Nicolas Menant

Stats

PHP VERSION	HOSTNAME	SERVER	PORT	CLIENT IP
5.6.26	bb48a55fe729	172.17.0.4:32772		10.1.20.4

If you remember, agent01 interface eth1 has the IP of 10.1.10.11

```
user@mesos-agent01:~/mycontainer$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 52:54:00:01:9e:de
          inet addr:10.1.20.101  Bcast:10.1.20.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fe01:9ede/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5099976 errors:0 dropped:2 overruns:0 frame:0
          TX packets:4173221 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1911678391 (1.9 GB)  TX bytes:2094211863 (2.0 GB)
```

Let's create a route on our windows client so that all traffic related to the container's network is sent to our server01 interface:

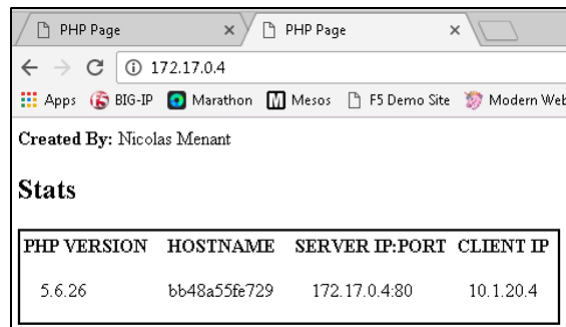
Open a **Windows terminal window** (you have a shortcut on your desktop)



In the windows terminal, run:

```
route add 172.17.0.0 mask 255.255.0.0 10.1.10.11
```

In Chrome open a tab to **http://[container ip]**



What happened? On server01 IP forwarding is enabled. When we created a static route from the Windows desktop to the Linux host we are able to forward packets directly to the Linux bridge network and by-pass the IPtables rules that were used previously for port forwarding.

You can check ip forwarding is enabled by running this command on **server01**

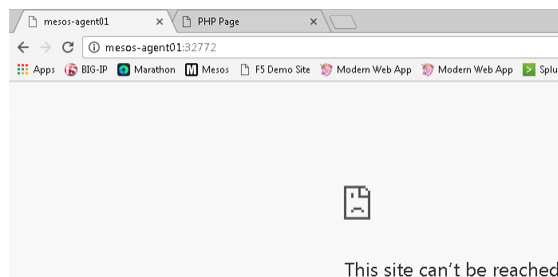
```
cat /proc/sys/net/ipv4/ip_forward
```

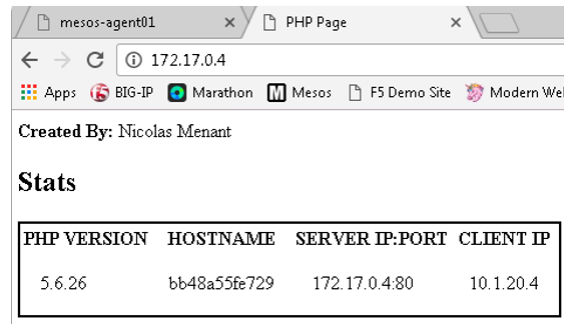
```
user@mesos-agent01:~/mycontainer$ cat /proc/sys/net/ipv4/ip_forward
1
```

Now run again:

```
docker restart myapp2
```

Reload both browser windows.





Created By: Nicolas Menant

**Stats**

PHP VERSION	HOSTNAME	SERVER IP:PORT	CLIENT IP
5.6.26	bb48a55fe729	172.17.0.4:80	10.1.20.4

Observe that you can no longer connect to using the previous port value, but can still connect via the linux bridge.

## 8.3 Docker and networking

As we have seen in previous lab, the networking setup of our containers are done automatically.

Docker provides a default network bridge and use it to attach containers to the network. This default network is 172.17.0.0/16 and leverage bridge0 interface. You can create your own bridge / network when needed.

If you want to review your bridge interface and the containers attached to it, you can do the following on **mesos-agent01**:

```
docker network ls
```

```
user@mesos-agent01:~/mycontainer$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1f443785159f        bridge             bridge              local
c1c3368c7fde        host               host                local
79a86d1ead7e        none              null                local
```

Here you can see the *bridge* network which is what is used by docker container by default. If you want to run a container in a specific network, you can use the `--network` option when using **docker run**

the *none* network adds a container to a container-specific network stack. That containers lacks network interface

The *host* network adds a container on the hosts network stack. You'll find the network configuration inside the container is identical to the host.

let's review what has been deployed over the bridge network. Copy the network ID for your bridge (in the previous screenshot, it is 1f443785159f)

```
docker network inspect *[NETWORK ID]*
```

Here you will see:

- the network configuration
- IPv4/v6 addresses that have been associated with each container

```
user@mesos-agent01:~/mycontainer$ docker network inspect 1f443785159f
[
  (
    "Name": "bridge",
    "Id": "1f443785159fe8fdab3b8502e860bd2f18e94170161e146fc638517483dbf9a1",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        (
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        )
      ]
    }
  )
]

"Containers": {
  "15221af326c5911611cb8231619b1b62a5f5b687": {
    "Name": "myapp",
    "EndpointID": "a3c21efed486fbe2681235",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  }
},
```

More advanced examples of Docker networking include *Docker Swarm* that utilizes its own SDN to provide multi-host Docker networking. The Kubernetes project utilizes flanneld for multi-host Docker networking that can leverage *host-gw* (basic L2/L3), UDP packet encapsulation, or VXLAN.

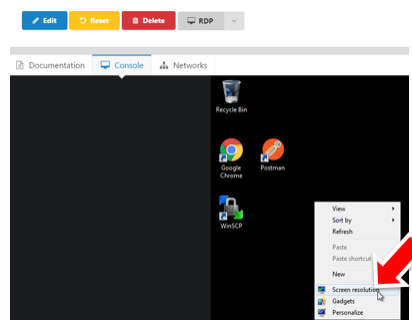




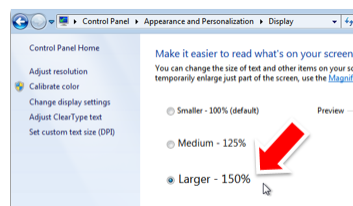
## 9.1 Changing display for HiRes displays

For HiRes Displays (Optional) If you find the text hard to read you may opt to change the resolution. You can either size your laptop display to something like 1920x1080

OR Login via the UDF Portal Console and change the display setting.



Click on *Make text and other items larger or smaller*



Logout the Windows client and reconnect via RDP.





## CHAPTER 10

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`