# Network softwarization Lab session 2: OS Virtualization Networking

Nicolas Herbaut David Bourasseau Daniel Negru

December 16, 2015

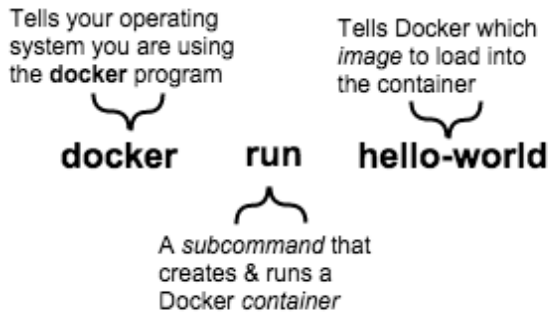
## 1 Introduction

### 1.1 Discovering docker

#### 1.1.1 Installation

Please launch the provided Virtual Machine with Virtual Box. Docker is already installed. Check that everything works with

```
$ docker run hello-world
```




### 1.2 Basic command line

This is a quick description of the docker command line. If you don't know docker, go through these explanations. You will use them in the rest of the lab.

**Download a pre-built image**

```
# Download an ubuntu image
$ docker pull ubuntu
```

**Running an interactive shell**

```
$ docker run -i -t ubuntu /bin/bash
```

The -i flag starts an interactive container. The -t flag creates a pseudo-TTY that attaches stdin and stdout.

To detach the tty without exiting the shell, use the escape sequence Ctrl-p + Ctrl-q. The container will continue to exist in a stopped state once exited. To list all containers, stopped and running, use the docker ps -a command.

**Selecting the name of the containers**

by default, containers have an ID (hex string) and an auto-generated name. You can force a particular name for convenience.

```
$ docker run --name containerX -i -t ubuntu /bin/bash
```

**Starting a long-running worker process**

```
# Start a very useful long-running process
$ JOB=$(docker run -d ubuntu /bin/sh -c "while␣true;\
␣do␣echo␣Hello␣world;␣sleep␣1;␣done")

# Collect the output of the job so far
$ docker logs $JOB

# Kill the job
$ docker kill $JOB
```

**Listing containers**

```
$ docker ps # Lists only running containers
$ docker ps -a # Lists all containers
```

**Controlling containers**

```
# Start a new container
$ JOB=$(docker run -d ubuntu /bin/sh -c "while␣true;␣\
do␣echo␣Hello␣world;␣sleep␣1;␣done")

# Stop the container
$ docker stop $JOB

# Start the container
$ docker start $JOB

# Restart the container
$ docker restart $JOB

# SIGKILL a container
$ docker kill $JOB

# Remove a container
```

```
$ docker stop $JOB # Container must be stopped to remove it
$ docker rm $JOB
```

**Running a command an existing container image**

```
$ docker exec -ti container_id /bin/sh
```

**attaching to a running container**

```
$ docker attach container_id
```

# 2 Docker Networking

## 2.1 Default Bridge Network

1. (1 point) Docker installed a Linux bridge docker0 in your system.
   (a) What is its mac address?
   (b) its IP address?
   (c) The subnet it belongs to?
2. (1 point) Inspect the bridge with the network command

   ```
   $ docker network inspect bridge
   ```

   By default, containers are attached to this default bridge. Launch 2 containers named container1 and container2 with an ubuntu image.
   (a) Can the two containers ping each other using their IPs?
   (b) Can the two containers ping each other using their host names (you can find the container's host names within the container by typing 'echo $HOSTNAME'?)
   (c) Give the commands or tools you used to find out.
3. (1 point) stop and delete container2. Relaunch it with with the following command.

   ```
   $ docker run -ti --name container2 --link container1 ubuntu
   ```

   (a) try to ping container1 from container2 using container1 hostname (eg.). What's the result?

   ```
   $container2 > ping container1
   ```

   (b) what mechanism does docker use to make container2 aware of container1?
   (c) what are the limitations of this technique?

### 2.2 Exposing ports

1. (3 points) launch the following command and explain what it does.
   (a)
   ```
   while true;do echo salut $HOSTNAME |nc -l 8080; done;
   ```
   (b) launch the following container image and connects on the 8080 port of the container.
   ```
   docker run --name container1 -d ubuntu /bin/bash -c 'while true;\
   do echo salut $HOSTNAME |nc -l 8080; done;'
   ```
   give the output of docker ps. From the host, can I establish a TCP connection to this container on port 8080? How?
   (c) launch the following container image,
   ```
   docker run -p 8080 --name container1 -d ubuntu /bin/bash -c \
   'while true; do echo salut $HOSTNAME |nc -l 8080; done;'
   ```
   give the output of docker ps. What's the difference? On which port of the localhost is the container reachable?
   (d) launch the following container image.
   ```
   docker run -p 8080:8080 --name container1 -d ubuntu /bin/bash -c \
   'while true; do echo salut $HOSTNAME |nc -l 8080; done;'
   ```
   give the output of docker ps. What's the difference? On which port of the localhost is the container reachable?
   (e) How does docker configure port forwarding? (Hint: you studied this tool in the very first lab session)

### 2.3 None Network

1. (½ point) what option of the "docker run" command allow specifying the network for the container?
2. (1 point) launch a container with the "none" configuration. Describe its network connectivity

### 2.4 Custom Bridged Network

It's possible to create isolated networks on which the containers will be attached.

```
$ docker network create --driver bridge network1
```

more info on the network can be retreive with the inspect command

```
$docker network inspect network1
[
    {
        "Name": "network1",
        "Id": "7009786ffb94f650eb1a71395cac3b28f68bbdcb9112662b6bedb0",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
```

```
            "Config": [
                {}
            ]
        },
        "Containers": {},
        "Options": {}
    }
]
```

1. (½ point) create 2 new bridged networks named network1 and network2. Create container1 on network1 and container2 on network2. Give the commands.
2. (3 points) (a) Can container1 access internet? Can container2 access internet?
   (b) Can container1 access container2?
   (c) What system tool (that you studied in this course) allows docker to enforce isolation of network1 and network2 as well as internet access for container
   (d) Show the configuration of this tool, explain the most important lines.

### 2.5 Host Network

1. (1 point) launch a container with the "host" configuration. Is the container network stack still segregated from the host networking? What are the issue with this configuration?

## 3 Build your own docker image

In this section, we explain briefly how to create docker images and run them. Go through section 3.1 to section 3.3, then answer the questions.

### 3.1 Write a Dockerfile

In this step, you use your favorite text editor to write a short Dockerfile. A Dockerfile describes the software that is "baked" into an image. It isn't just ingredients tho, it can tell the software what environment to use or what commands to run. Your recipe is going to be very short.

- Go back to your terminal window.
- Make a new directory by typing mkdir mydockerbuild and pressing RETURN.

  ```
  mkdir mydockerbuild
  ```

  This directory serves as the "context" for your build. The context just means it contains all the things you need to build your image.
- Change to your new directory.

  ```
  $ cd mydockerbuild
  ```

  Right now the directory is empty.
- Create a text file called Dockerfile in your current directory. You can use emacs, vi or leafpad (graphical) to do this.
- Open your Dockerfile file.

- Add a line to the file like this:

```
FROM ubuntu
```

The FROM keyword tells Docker which image your image is based on. Here you start from an ubuntu distribution.

- Now, add the fortunes program to the image.

```
RUN apt-get -y update && apt-get install -y iperf
```

iperf is a program that compute the available bandwidth between a client and a server. By default, it uses TCP.

- Once the image has the software it needs, you instruct the software to run when the image is loaded.

```
CMD iperf -s
```

- Check your work, your file should look like this:

```
FROM ubuntu
RUN apt-get -y update && apt-get install -y iperf
CMD iperf -s
```

- Save and close your Dockerfile

At this point, you have all your software ingredients and behaviors described in a Dockerfile. You are ready to build a new image.

### 3.2 Build an image from your Dockerfile

- Now, build your new image by typing the docker build -t iperf-server . command in your terminal (don't forget the . period).

```
docker build -t iperf-server .
Sending build context to Docker daemon 1.253 MB
Step 1 : FROM ubuntu
 ---> 89d5d8e8bafb
Step 2 : RUN apt-get -y update && apt-get install -y iperf
 ---> Using cache
 ---> ec78def14644
Step 3 : CMD iperf -s
 ---> Using cache
 ---> cf4bc851405c
Successfully built cf4bc851405c
```

The command takes several seconds to run and reports its outcome.

### 3.3 Run your new iperf-server

In this step, you verify the new images is on your computer and then you run your new image.

- If it isn't already there, place your cursor at the prompt in your terminal window.

- Type docker images and press RETURN.

```
docker images
REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE
iperf-server latest cf4bc851405c 3 minutes ago 210.4 MB
ubuntu latest 89d5d8e8bafb 7 days ago 187.9 MB
```

- Run your new image by typing docker run -ti iperf-server and pressing RETURN.

```
docker run -ti iperf-server
```

### 3.4 Using docker images

**Hint**

It's possible to pass variable from your host to the image with the -e option, for example:

```
$ docker run -ti -e FOO=BAR ubuntu
$ root@5b8fad1552de:/# echo $FOO
$ BAR
```

**Hint**

You can code a if statement in bash on one line

```
a=1 ; b=2 ; if [ $a -eq $b ] ; then echo "equal" ; else echo "not␣equal" ; fi
```

1. (1 point) launch an iperf-server docker image.
   (a) what is the IP of the image?
   (b) checkout iperf man page and run an iperf client from your host. What is the result of the test?
2. (1 point) compare the performances of the iperf-server docker image with its native counterpart. Is there a difference? what could explain it?
3. (2 points) create 2 dockerfiles in two different folders. One is an iperf-server, the other is an iperf-client that connects to the server. You should be able to run them as follow.

```
$ docker run -d -e MODE="tcp" --name perfsvr iperf-server
$ docker run -ti -e MODE="tcp" --link perfsvr iperf-client
```

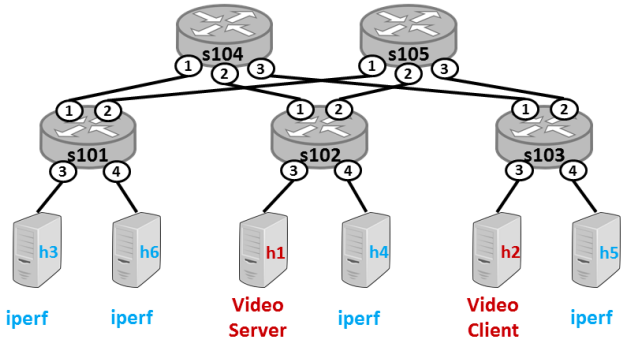
   (a) give the dockerfile for the server
   (b) give the dockerfile for the client
4. (1 point) compare the iperf UDP performances when (1) both iperf server and client being run on the host (2) both iperf server and client being run from containers.

# 4 OpenVSwitch

In this section, we learn how to program openvswitch using the command line. Mininet uses openvswitch to handle l2 and l3 routing.

For this section, **login as the root user** (sudo su) and type

```
echo "alias␣ovs-ofctl='ovs-ofctl␣-O␣OpenFlow13'" >> ~/.bashrc
source ~/.bashrc
```




1. (3 points) launch td01's mininet dash topology. We will not use ryu this time, so press enter without starting ryu.

   ```
   /home/re350/re350/td01/dash/mdc --vid
   ```

   (a) can we see the video? explain why

   (b) on top of mininet, you can display the topology with openvswitch

   ```
   ovs-vsctl show
   ```

   unbind the controller from each bridge using

   ```
   ovs-vsctl del-controller <<switch_name>>
   ```

   now we are going to set each ovs instance as a learning switch

   ```
   ovs-ofctl add-flow <<switch_name>> actions=normal
   ```

   Can you access the video? what is the issue if every switch is connected in the network?

   (c) We can solve this issue by destroying some link

   ```
   ovs-vsctl del-port <<port_name>>
   ```

   (Hint: you are doing the job of the ST protocol)
   Give the command you issued to make the network usable and the video available.

   (d) what is the option allowing ovs-vsctl to remove a bridge?

   (e) remove s101 bridge. What's the impact on the video quality? explain why.

   (f) remove every ports linking iperf host to aggregation switches. What's the impact on the video quality? explain why.

# 5 Bonus OVS+DOCKER

- Create 1 container image that exposes ports 8080 and 8081 (see Section 2.2).
- connect 3 instances (D1, D2, D3) of this image to an openvswitch (ovs-docker)
- implement the following rules using openflow.
  - D1, D2 and D3 can ping each other
  - D1 can create a TCP connection to D2's 8080 port
  - D3 can create a TCP connection to D2's 8081 port
  - no other TCP communication should be

1. (5 points) give the ovs-ofctl commands you used to configure the openvswitch.

### Hint

The following command creates the new interface *eth0* in the container *container1* with the ip *10.10.10.10* on the open vswitch bridge named *ovs-br*.

```
ovs-docker add-port ovs-br eth0 container1 --ipaddress=10.10.10.10/24
```

### Hint

some usefull ovs-ofctl commands:

```
#switch acts as a learning switch for arp queries
ovs-ofctl add-flow ovs-br0 arp,actions=normal

#switch acts as a learning switch for icmp queries
ovs-ofctl add-flow ovs-br0 icmp,actions=normal

#drop traffic from D2 to D1 on tcp for port 9999
ovs-ofctl add-flow ovs-br0 ip,tcp,tcp_dst=9999,actions=drop

#display port number of a bridge
ovs-ofctl add-flow ovs-br0 show <<bridge_name>>

#send tcp packet arriving to port 1 to port 2
ovs-ofctl add-flow ovs-br0 ip,tcp,in_port=1,actions=output:2
```