Workshop : Docker and Open vSwitch

Abstract

In this workshop, you will get introduced to Docker and Open vSwitch as the fundamental technologies for creating virtual network topologies. Sections 2 and 3 provide relevant background and usage respectively of Docker and Open vSwitch. An experienced student may skip those and jump straight to Section 4 which describes the tasks to be completed in the workshop.

1 Expected outcome

In this workshop, the student would learn about:

1. Using Docker as virtualization tool to create virtual hosts
2. Using Open vSwitch to create a virtual network topology
3. Connect the virtual hosts to virtual switches and test connectivity

This workshop and all subsequent ones need to be implemented on a Linux host. All workshops have been tested on an Ubuntu 16.04 Virtual Machine on VirtualBox.

2 Background

This section provides a basic background about Docker and Open vSwitch.

2.1 Docker

Docker is essentially operating-system-level virtualization which allows developing and deploying software in packages called *containers*. Docker containers are isolated from each other and bundle their own software, libraries and configuration files - and therefore are platform independent. Containers use the kernel of the host machine, and are therefore much more lightweight compared to virtual machines.

A container is deployed using an *image* that specifies the details of the container. An image is usually built by extending a base image (e.g. ubuntu-trusty) with specific configurations.

Install Docker on the Linux host by following the instructions in the following link : How To Install and Use Docker on Ubuntu 16.04 [1].

Upon installing Docker, you need to start Docker service by using the following command.

```
$ sudo systemctl start docker
```

2.2 Open vSwitch

Open vSwitch is a multilayer software switch licensed under the open source Apache 2 license. The software aims to provide a production quality switch platform that supports standard management interfaces and opens the forwarding functions to programmatic extension and control.

Install Open vSwitch on the Linux host using the following command.

```
$ sudo apt-get install bridge-utils openvswitch-switch
```

Upon installing Open vSwitch, you need to start the Open vSwitch service by using the following command.

```
$ sudo /usr/share/openvswitch/scripts/ovs-ctl start
```

3 Basic functionalities

This section describes the basic command-line tools for using Docker and Open vSwitch for the purpose of this workshop.

[1] https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04

3.1 Running a vanilla Docker container

Execute the following command to launch a Docker container.

```
$ docker run -it ubuntu:trusty sh
```

Through this command, we have instructed the Docker runtime to

1. Launch a container based on the "ubuntu:trusty" image
2. Launch the "sh" terminal in the container
3. Attach to an interactive tty in the container using the "-it" flag

Exiting from the terminal will kill the "sh" shell - which is the main process of the container. This will lead to the container being killed by the Docker runtime.

3.2 Creating a custom Docker image

Now we will extend the base "ubuntu" image to include additional tools. The recipe of a Docker image is contained in a Dockerfile, whose contents are shown below.

```
# Specify the base image that this image will be based on
FROM ubuntu:trusty

# Use the RUN keyword to specify specific changes to be made to the base image
RUN apt-get update && apt-get install -y iperf3
RUN apt-get install -y tcpdump && mv /usr/sbin/tcpdump /usr/bin/tcpdump
RUN apt-get install -y hping3
```

Now create an image named "endpoint" using created Dockerfile. You will need to navigate to the directory containing the Dockerfile before executing the following command.

```
$ docker build -t endpoint:latest .
```

3.3 Executing a command on running container

The following command shows how to generate 3 ICMP requests to the host 192.168.1.2 from the container "h1".

```
$ docker exec -it h1 ping -c 3 192.168.1.2
```

The options "-it" are used in order to have the command executed interactively instead of in the background. This command can be used to run a single command on a running Docker container. For more information check out this page : docker exec[2]. You could also do the following in order to launch a shell on a container.

```
$ docker exec -it h1 bash
```

The above command will open a new bash shell in the container, which can be used to execute commands on the container just as you would on a bare-metal machine.

3.4 Creating a virtual switch

In the following command we use Open vSwitch's configuration utility to create a new virtual switch (also called bridge).

```
$ sudo ovs-vsctl add-br ovs-br1
```

3.5 Connecting a Docker container to virtual switch

In the following command we add a port "eth0" on container "cnt" with an IP address "192.168.1.2/24" that connects to virtual switch "ovs-br1". When adding interfaces to containers, take a note of the interface names as they will be used later in the workshop.

```
$ sudo ovs-docker add-port ovs-br1 eth0 cnt --ipaddress="192.168.1.2/24"
```

4 Workshop Specification

In this workshop you are required to implement the network topology shown in Figure 1. You can refer to Section 3 for hints on commands used to complete the required tasks. The target topology consists of two Docker containers connected to each other using an Open vSwitch. First, you need to create a Docker image using the Dockerfile at this link[3]. Use the tag "endpoint:latest" for the created image.

Then you need to create containers using that image. The following command shows how to run a container with specific arguments suited for this workshop.

[2] https://docs.docker.com/engine/reference/commandline/exec/
[3] https://github.com/harshitgupta1337/sicc_nfv/blob/master/workshop_1/Dockerfile

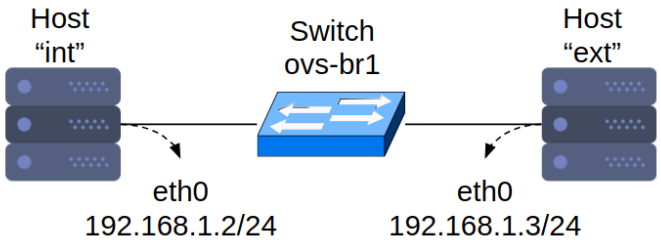

Figure 1: Schematic of network topology for this exercise.

```
$ docker run -d --privileged --name=int --net=none endpoint:latest tail -f /dev/null
```

1. The default networking needs to be disabled in Docker containers using the ”–net=none” argument.
2. The container should be run with the ”–privileged” flag
3. The ”-d” flag makes the container run as a daemon
4. The container should be alive for the duration of the experiment. Hence the container is instructed to follow the contents of the file /dev/null, which effectively makes it stay alive following an empty file forever.

Upon setting up the network topology you need to verify that the Docker containers are able to communicate with each other through the switch.

5 Requirements

Verify that the following requirements are met.

- The topology needs to correctly implemented. Both the Docker containers should be able to ping (ICMP) each other.
- Use the *nc* command-line tool (usage of *nc* can be found in Section 7) to generate TCP traffic to a specific port. Use *tcpdump* on network interfaces on both Docker containers to record packets being sent from and received on that interface.

  Note that you would need to launch *nc* and *tcpdump* on multiple terminals to listen on multiple interfaces. For example, one terminal runs *tcpdump* to record packets sent/received on interface *eth0* of container *int*, and another terminal runs *tcpdump* to record packets sent/received on interface *eth0* of container *ext*.

6 Deliverables

- An executable script ”*setup_topology.sh*” that performs all actions needed for setting up the topology for this workshop. The script should be complete, in that after executing the script, the containers should be reachable from each other.
- Show a demo of the working topology to the TA.
- A document containing the packet traces from tcpdump for the TCP traffic test explained in Section 5. The document should mention the various types of protocols seen in the packet trace and the role of each protocol. Special attention should be given to ARP packets as ARP will be used in the next workshops.

7 Useful References

7.1 Usage of *nc*

1. Start listening on the destination port (port 80 in this example)

   ```
   $ nc -l 80
   ```

2. Initiate connection to the destination port (destination host IP is 192.168.1.2)

   ```
   $ nc 192.168.1.2 80
   ```

3. Send data on the TCP connection by typing plaintext
4. Terminate the TCP connection by hitting Ctrl+C on either terminals

7.2 Other useful references

- How to create Docker Images with a Dockerfile[4]
- A tcpdump Tutorial with Examples — 50 Ways to Isolate Traffic[5]

[4] https://www.howtoforge.com/tutorial/how-to-create-docker-images-with-dockerfile/
[5] https://danielmiessler.com/study/tcpdump/