Lab 8. NFV Lab based on Docker (Ryu Controller)

Contents

Experiment Task Design

This lab aims to let students understand and integrate OVS-Switch with Docker containers in CloudLab. This lab uses Ryu Controller, one of the open-source SDN controllers. With this understanding, students can create SDN-based docker network and control using SDN Controller.

Submission

Take screenshots of all the steps involved and explain in one or two paragraphs. Study the flows rules below and explain the meaning of the printed flow rules on the terminal.

Students can refer the link (http://docs.cloudlab.us/cloudlab-tutorial.html) for more details about creating profiles on CloudLab. Students should have an account of either CloudLab or GENI or any other federated services like EmuLab to access CloudLab. CloudLab login page: https://www.cloudlab.us/login.php

Profile Setup

Create a profile for an SDN controller

Students create a profile with a single node as shown in Figure 1. Use '**Ubuntu 18**' as the OS and select "any" for the hardware type. Accept the topology and then click on the ***create*** button. Please give appropriate descriptions and then instantiate the experiment by clicking on the ***instantiate*** button. You will be asked to select a cluster. Select the available cluster and then click the "***Finish***" button. Once the experiment is online, students need to install Ryu controller on the node.

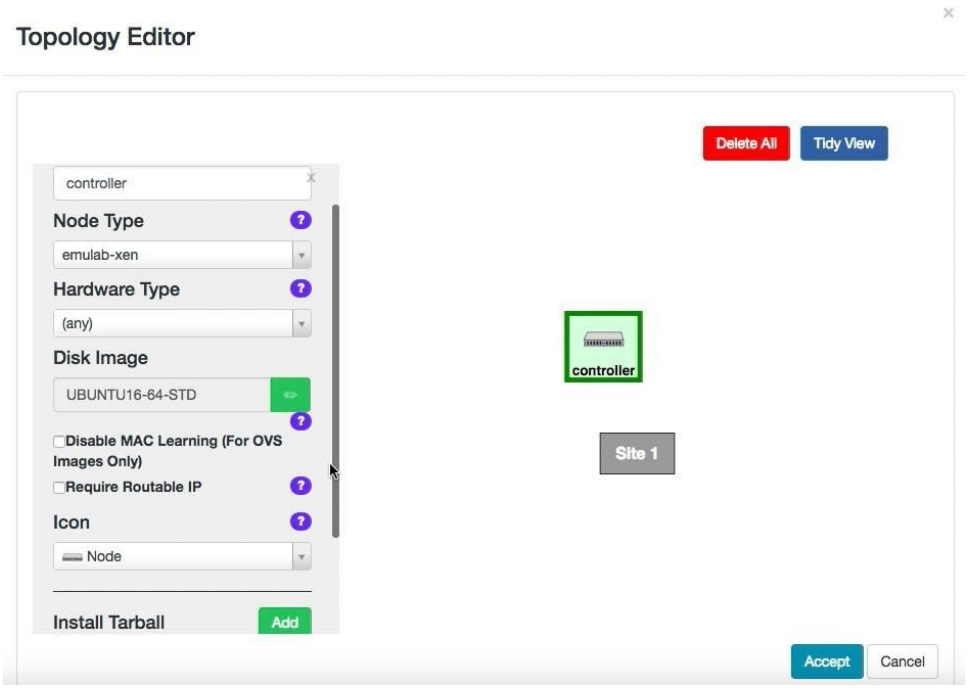


Figure 1: Setup a single node for an SDN controller. (Note that if you check "Require Routable IP," the controller can be assigned with a public IP address that other nodes can access. If not, the controller will have a private IP address.)

Installing Ryu Controller

1) Open a new terminal.

2) Run "**ifconfig**" and note down the IP address. This IP address will be used whenever the topology is needed to being setup for an experiment.

3) To install Ryu controller, perform the following steps:

Get sudo user privileges

sudo su

Update APT repo

apt-get update

Download Ryu repo from Github

git clone https://github.com/osrg/ryu.git

Install dependencies

apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev

apt install python3-pip

Install Ryu Controller:

cd ryu

pip3 install -r tools/pip-requires

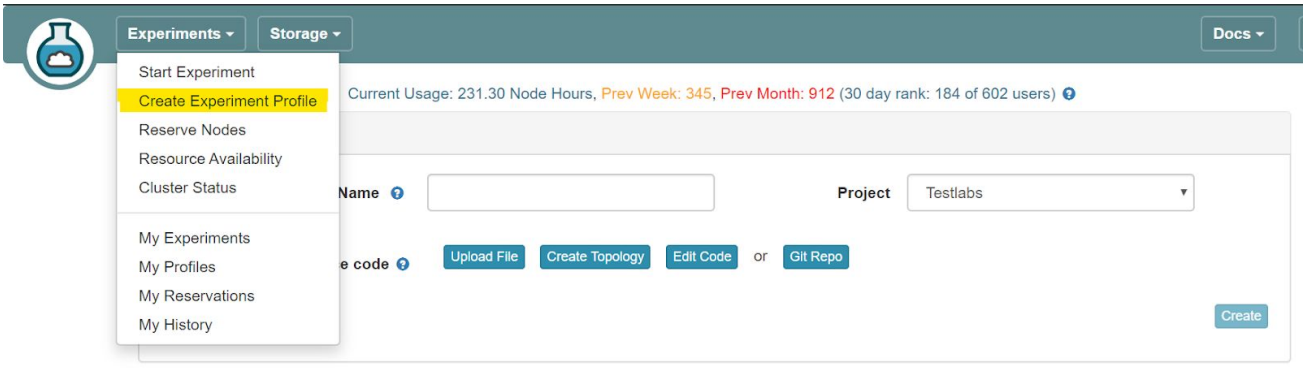
python3 setup.py install

(Notes) You can use "sudo /bin/sh set_ryu.sh" run all those commands to save your installation time. After installation, run "cd " to switch to the destination directory.

4) Start the controller: "**ryu-manager --verbose yourapp.py**".
The "verbose" option prints the debug output. All the default applications are in "ryu/app/" directory. For example, "**ryu-manager –verbose ryu/app/simple_switch_13.py**"

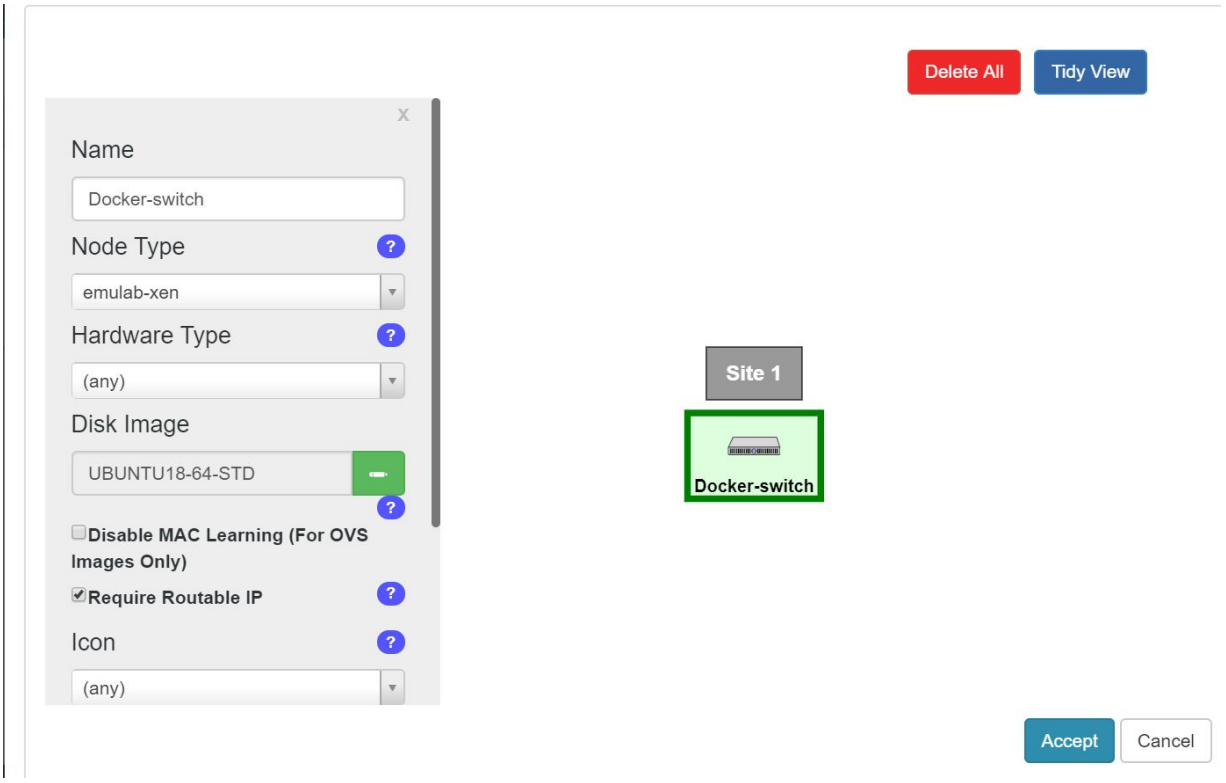
Setup a topology profile for the experiment

1) Create a profile for the experiment using "Create Experiment Profile" in "Experiments" tab.



2) Upload the profile.xml file using the "Upload File" option. You can give any name to the profile and create it by clicking on "create" button at the bottom of the page. This creates a profile with single node with routable IP and the following configuration. You can view it using "Edit Topology" button.

OS: Ubuntu 18
Node Type: emulab-xen
Memory: 4 GB
CPU: 2



3) Click on "instantiate" button to start this node. Give a name to the experiment and select "Emulab" as the cluster. Continue to click "Next" to complete the instantiation of the node.

Conducting the Lab

Install Open vSwitch and Docker on the node.

1) Run "**sudo apt-get update"** and **"sudo apt-get install openvswitch-switch**" to install OpenVSwitch.

2) Run "**sudo apt-get install docker.io -y**" to install docker.

3) Use the following commands to create a bridge and join it to the controller.

```
sudo su
ovs-vsctl add-br <bridge_name>
ovs-vsctl set-controller <bridge_name> tcp:<controller_IP_Address>:6653
```

bridge_name can be any name of your interest and controller_IP_address is the eth0 address on the controller node created previously.
The above commands create a bridge and can be verified using the command : "**sudo ovs-vsctl show**"

```
root@docker-switch:~# ovs-vsctl add-br ovs-lan
root@docker-switch:~# ovs-vsctl set-controller ovs-lan tcp:155.98.37.91:6653
root@docker-switch:~# ovs-vsctl show
adb6b597-8da1-44c3-af07-2d884a6fc763
    Bridge ovs-lan
        Controller "tcp:155.98.37.91:6653"
        Port ovs-lan
            Interface ovs-lan
                type: internal
    ovs_version: "2.9.5"
root@docker-switch:~#
```

4) Now, pull the docker image on to the node using the command "**docker pull gantapritham4/docker:latest"**.
   This command pulls the docker image from docker hub repository to the local storage. This image is a basic docker container with Ubuntu OS.
   The list of docker images in local storage can be checked by the command "**sudo docker images**"

5) Open a new terminal and run the command "**sudo docker run -it --name docker1 gantapritham4/docker /bin/bash**"
   The command "docker run" starts a container and argument "-it" opens a terminal to the container. The option "--name" assigns a name to the container. "gantapritham4/docker" is the name of the docker image to be started and "/bin/bash" is the process to be started inside the container once it starts running.

6) Open a new terminal and repeat the command to run a second container with a different name.(Example "**sudo docker run -it --name docker2 gantapritham4/docker /bin/bash**")

   Use the command "**docker ps -a**" on the host to show the list of all running containers.

7) Once we have two docker containers, these two dockers should be connected to the ovs-switch created. It can be done using the following command:

   ovs-docker add-port <switch_name> <interface_name> <container_name> --ipaddress=<ip_address>

   Note the name of the new interface created on the host by using the command "ifconfig -a". The name of the interface created will be random and end with "_l". This helps in understanding the flow rules.

   For example: ovs-docker add-port ovs-lan eth1 docker1 --ipaddress=10.0.0.1/24
   ovs-docker add-port ovs-lan eth1 docker2 --ipaddress=10.0.0.2/24

   NOTE: Interfaces should be added starting from eth1 because every container starts with a default interface eth0

8) Port addition can be validated both inside the container and the host. Inside the container, run the command "**ifconfig -a**" to list the interfaces and their IP addresses. On the host, run the command "**sudo ovs-vsctl show**" to show the info on ovs-switch.

```
adb6b597-8da1-44c3-af07-2d884a6fc763
    Bridge ovs-lan
        Controller "tcp:155.98.37.91:6653"
        Port ovs-lan
            Interface ovs-lan
                type: internal
        Port "5050cb36ede54_l"
            Interface "5050cb36ede54_l"
        Port "03280a50cddc4_l"
            Interface "03280a50cddc4_l"
    ovs_version: "2.9.5"
root@docker-switch:~#
```

The above screenshot shows two interfaces added to the ovs-switch for both the containers.

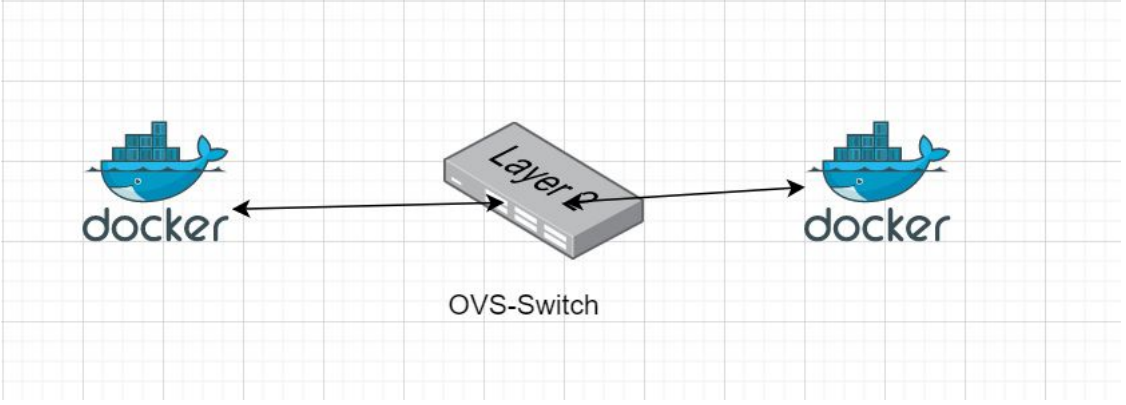
```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 0.0.0.0
        ether de:bc:4a:59:6f:3d  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 866 (866.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@356cf8989e37:/#
```

The screenshot above is from the container which shows the interface eth1 and its assigned IP address 10.0.0.1.

The topology looks like the image shown below.



Connectivity Test between 2 Dockers

Run "**apt-get install iputils-ping**" to install the ping program in both the containers. Run "ping 10.0.0.2" on docker1 container and check the response. Ping can be tested from either of the containers.

Ping and dump flows

Once all the configuration settings are finished, the ***Ping*** command from ***docker1*** to ***docker2*** will start working. Students can use "***tcpdump -i eth1***" on ***docker1*** and ***docker2*** to check the responses for the ***Ping*** command. In addition, student can check the flow rules on the host nodes using the following command, "***ovs-ofctl dump-flows <bridge_name> -O OpenFlow13***" by replacing <bridge_name> with the configured bridge names for each node.

```
root@docker-switch:~# ovs-ofctl dump-flows ovs-lan -O OpenFlow13
 cookie=0x0, duration=799.520s, table=0, n_packets=29, n_bytes=2562, priority=1,in_port="5050cb36ede54_l",dl_src=fa:a9:e6:79:a1:1e,d
l_dst=de:bc:4a:59:6f:3d actions=output:"03280a50cddc4_l"
 cookie=0x0, duration=799.515s, table=0, n_packets=28, n_bytes=2464, priority=1,in_port="03280a50cddc4_l",dl_src=de:bc:4a:59:6f:3d,d
l_dst=fa:a9:e6:79:a1:1e actions=output:"5050cb36ede54_l"
 cookie=0x0, duration=871.021s, table=0, n_packets=3, n_bytes=182, priority=0 actions=CONTROLLER:65535
root@docker-switch:~#
```

Figure 5: Flow rules inserted by the SDN controller

Appendix

OVS commands:

ovs-vsctl: Used to configure the ovs-vswitchd configuration database (known as ovs-db)

Example: To delete a bridge: "**ovs-vsctl del-br ovs-lan1**"

ovs-ofctl: A command line tool to monitor and control OpenFlow switches

Example: To print the OVS flow rules "**ovs-ofctl dump-flows ovs-lan2 -O OpenFlow13**"

sudo docker stop <container_name> : To stop the container

sudo docker rm <container_name>: To completely remove the container image from the host.

sudo docker ps -a : Lists all the running and stopped containers.