Hands-on Lab Description

CS-NET-00009 – Containernet Lab

Category:

```
CS-NET: Computer Networking
```

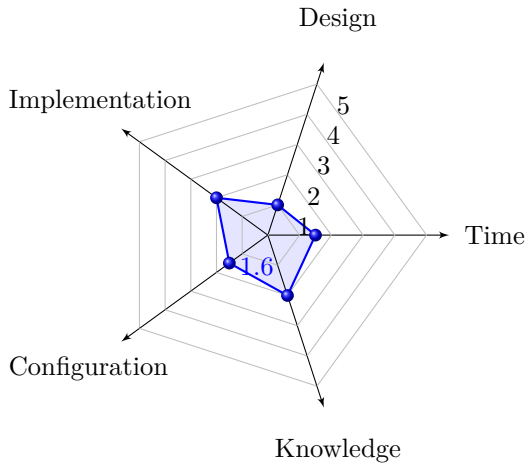
Objectives:

```
1 Learn to deploy the containers using containernet on virtual machine
2 Learn the functionality of the Containernet
```

Estimated Lab Duration:

```
1 Expert: 20 minutes
2 Novice: 55 minutes
```

Difficulty Diagram:



Difficulty Table.

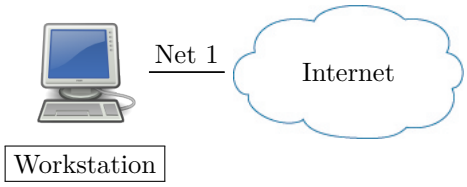
Measurements	Values (0-5)
Time	1.5
Design	1
Implementation	2
Configuration	1.5
Knowledge	2
Score (Average)	1.6

Required OS:

```
Linux: Ubuntu 18.04 LTS (Xenial Xerus)
```

Lab Running Environment:

```
VirtualBox https://www.virtualbox.org/ (Reference Labs: CS-SYS-00101)
```




```
1  Workstations: Linux (Ubuntu 18.04 LTS)
2  Internet: Accessible
```

Lab Preparations:

```
1  Open Virtual Switch (OVS) (Reference Lab: CS-NET-00006)
2  Mininet (Reference Lab: CS-NET-00007)
3  POX controller (Reference Lab: CS-NET-00008)
```

Task 1.0 Initial setup and connectivity testing

In this task, you will start to deploy the Containernet in the Thoth VMs, the Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network typologies. This enables interesting functionalities to build networking/cloud emulators and testbeds.

1. You can install the Containernet by Bare-metal installation on Ubuntu 18.04, and the Docker and Mininet will be installed along with the Containernet. Note that dependency errors may occur due to some previously installed packages, and if you cannot solve it, a vm rebuild may be needed, and you can start the following installation from a new VM that may take a few hours to finish.

```
$       sudo apt update
$       sudo apt-get install ansible git aptitude
$       git clone https://github.com/containernet/containernet.git
$       cd containernet/ansible
$       sudo ansible-playbook -i "localhost," -c local install.yml
```

2. Additionally, it is important to have the net-tools and ansible installed in your system. If they are already installed, you can move to the next step.

```
$       sudo apt install net-tools % install net-tools
```

3. You can verify if Conainernet is installed successfully by running the example script under directory of the *containernet*:

```
$       sudo python3 examples/containernet_example.py
$       exit
```

4. Then, you can verify the Mininet installation by send test ping command to a temporary topology

```
$       sudo mn --test pingall
```

Task 2.0 Customize the topology using the Containernet

In this task, you will create a simple network topology by executing a Python script.

1. After you successfully installed the Containernet, there will be several example scripts built for creating the sample container typologies. Next, you will get familiar with a basic topology consist of two hosts and switches.
   Note: all the operations such as adding nodes are done by Python-based scripts.

   Create a new script called *containernet_example.py* under directory of Containernet and import the necessary python packages from the *mininet* folder.

```
from mininet.net import Containernet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import info, setLogLevel
setLogLevel('info')
```

This part of code is used to display the information and add a kernel controller to our topology. Beside, you are assigning the IP address *10.0.0.251* and *10.0.0.252* to host1 and host2, respectively, with an standardized image *ubuntu:trusty* and you can change these parameters as you want.

```
net = Containernet(controller=Controller)
info('*** Adding controller\n')
net.addController('c0')
info('*** Adding docker containers\n')
d1 = net.addDocker('d1', ip='10.0.0.251', dimage="ubuntu:trusty")
d2 = net.addDocker('d2', ip='10.0.0.252', dimage="ubuntu:trusty")
```

Next, you are adding the switches and link the host1 to switch1 and host2 to switch2.

```
info('*** Adding switches\n')
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
info('*** Creating links\n')
net.addLink(d1, s1)
net.addLink(s2, d2)
```

Next, you will define a function to start the mininet topology

```
info('*** Starting network\n')
net.start()
```

You can pre-identify set a ping request while starting the topology

```
info('*** Testing connectivity\n')
net.ping([d1, d2])
```

Starting the CLI (Computer Telephony Integration) in the end

```
CLI(net)
```

Stop the network when catch the stop request

```
info('*** Stopping network')
net.stop()
```

You can run the script to start our network toplogy

```
$       sudo python3 containernet_example.py
```

2. As you can observe the output after running the $containernet_example.py$ above, the container $d1$ is not able to ping container $d2$ because there isn't a link between each other. However, the $d1$ and $d2$ are connected to $s1$ and $s2$ individually, so you want to connect to $s1$ and $s2$ to enable the communication of $d1$ and $d2$
Next, please add a TCLink beetween two switches and set the delay time in 100ms and bandwidth in 1. It is important to show $d1$ can reach $d2$ such as sending several ping packets from one to another.

Related Information and Resource

Other frequently used Docker and Containernet command

1. List all built docker images

```
$       sudo docker image ls
```

2. Delete all docker images in the disk

```
$       sudo docker system prune -a
```

3. Clean up and mininet caches

```
$       sudo mn -c
```

Lab Assessments (100 points)

Lab assessment for accomplishing Task 1, Task 2, and Task3 depends on the following facts:

1. (35 points) The containernet topology can
   - successfully implement the test ping command in a temporary topology

```
        sudo mn test --pingall
```

2. (65 points) After started the scripts the *container_example.py* can
   - create the indicated topology

```
$         python3 container_example.py
```

```
        *** Adding controller
        *** Adding docker containers
        1:
        ubuntu; trusty; None;
            sha256:04cc778d3901e3dd520686c6a30a306fa39c238c001321cdf5035dbafec591c9
        d1: kwargs {'ip': '10.0.0.251'}
        d1: update resources {'cpu_quota': -1}
        1:
        ubuntu; trusty; None;
            sha256:04cc778d3901e3dd520686c6a30a306fa39c238c001321cdf5035dbafec591c9
        d2: kwargs {'ip': '10.0.0.252'}
        d2: update resources {'cpu_quota': -1}
        *** Adding switches
        *** Creating links
        (1.00Mbit 100ms delay) (1.00Mbit 100ms delay) (1.00Mbit 100ms
            delay) (1.00Mbit 100ms delay) *** Starting network
        *** Configuring hosts
```

```
d1 d2
*** Starting controller
c0
*** Starting 2 switches
s1 (1.00Mbit 100ms delay) s2 (1.00Mbit 100ms delay) ...(1.00Mbit
    100ms delay) (1.00Mbit 100ms delay)
```

- enable the communication between the $d1$ and $d2$ (link up the two switches)

```
containernet> d1 ping d2
```

The challenging task evaluation is optional and the instructor can decide on how to evaluate students' performance.