OpenvSwitch and VNF setup - Host and SDN Controller

1. Main Test topology:

Five test VMs are installed on the Dell Server using OpenStack Orchestrator or using KVM/virt-manager. Two VMs will serve as the primary and secondary SDN Controllers, two VMs will serve as host1 and host2, and one VM will be used for testing the setup as shown in the diagram (Figure 1) as shown below:

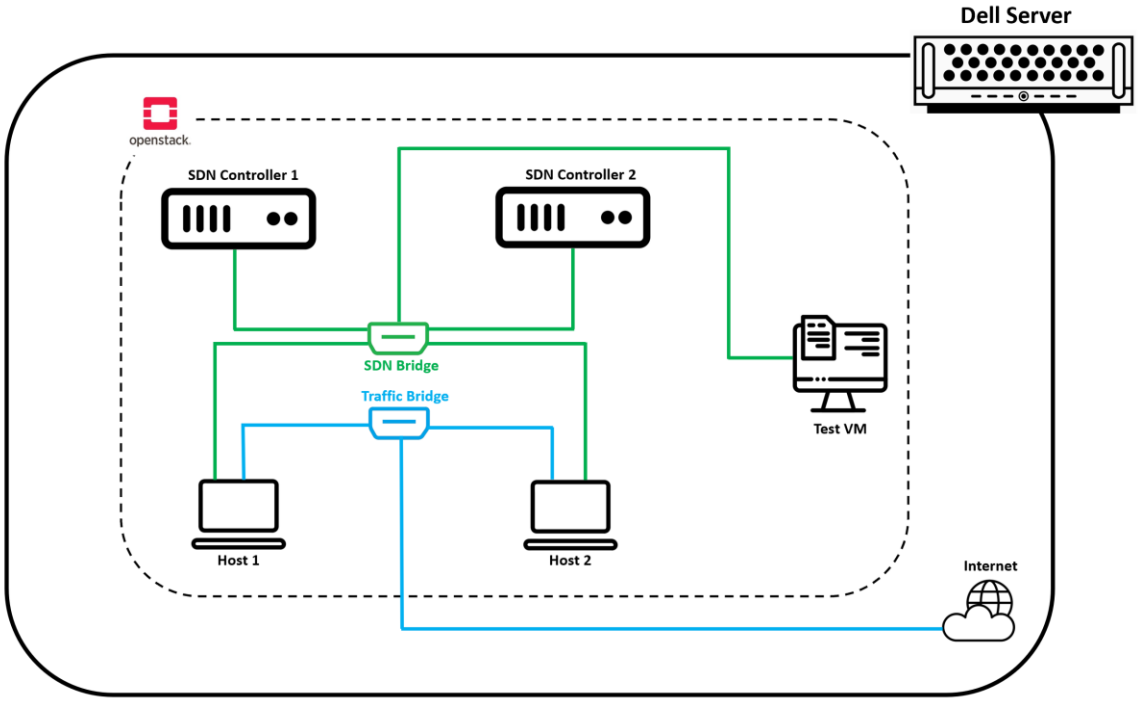

Figure 1: Capstone project setup topology diagram

The VMs are connected to each other using the SDN bridge created on the Dell Server. The host devices are connected to each other and to the internet via Traffic bridge. All the traffic is traversed using this bridge, as shown in the diagram above.

2. Host VM setup guide:

VNFs are deployed on the host devices using containerized OpenvSwitches. Each flow table in an OpenvSwitch act as an individual VNF. Linux bridges are used to interconnect the switches to each other as well as to the host interface and traffic diversion. The traffic is diverted into the VNFs using bridge ovs-br1 through policy-based routing. Traffic is forced to traverse through multiple VNFs such as firewall, router, switch, QoS, and VLAN. The service chain is created using multiple VNF combination. The output of these service chains will drop/modify/forward the traffic according to the flow table rules. The traffic after traversing through the VNFs is forwarded to the exit traffic interface by ovs-br3, as shown in the Host VM setup diagram below (Figure 2).

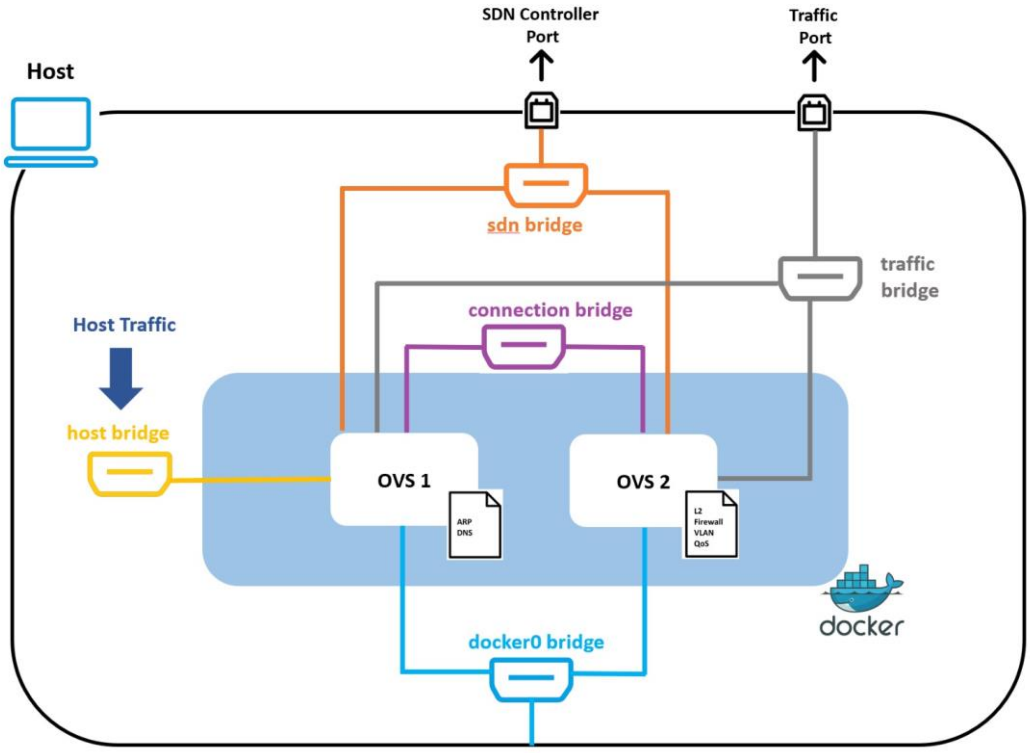

Figure 2: Host VM setup diagram

A. Docker OpenvSwitch installation:

Two OpenvSwitches are used to set up the VNFs on the host device to create service chains for ensuring an end to end network performance. globocom/openvswitch image is used to initialize docker based switches. Below are the commands used to initialize vswitch1 and vswitch2.

```
# docker run -i -t --rm --name vswitch1 --cap-add=NET_ADMIN -d
globocom/openvswitch
```

```
# docker run -i -t --rm --name vswitch2 --cap-add=NET_ADMIN -d
globocom/openvswitch
```

The successful implementation will give an output like below:

```
[root@localhost ~]# docker run -it --rm --name vswitch --cap-add=NET_ADMIN -d globocom/openvswitch
4a8520d6fd08c8452bfa1f8391b4ba4ca904e9e340794c5f4daae3af750c4759
[root@localhost ~]#
[root@localhost ~]# docker run -it --rm --name vswitch1 --cap-add=NET_ADMIN -d globocom/openvswitch
bb88a33de9844a2180c5c165a94bd82a722a5e6e6e3fc744c180d599b60cde38
[root@localhost ~]#
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS               NAMES
4a8520d6fd08        globocom/openvswitch   "/bin/sh -c /usr/b..."   About a minute ago  Up About a minute   22/tcp              vswitch
bb88a33de984        globocom/openvswitch   "/bin/sh -c /usr/b..."   3 minutes ago       Up 3 minutes        22/tcp              vswitch1
[root@localhost ~]#
```

B. Linux Bridge Creation and Connection:

Create 4 linux bridges- br0 will force the traffic into the VNFs, ovs-conn will connect the switches, ovs-traffic will connect the switches to the exit traffic interface and ovs-sdn will connect the openvSwitches to the SDN Controller. The commands and output for the bridge creation are shown below:

```
#  ovs-vsctl add-br br0
#  ovs-vsctl add-br br-sdn
#  ovs-vsctl add-br br-traffic
#  ovs-vsctl add-br br-conn
```

Connect the switch interfaces using following commands:

```
ifconfig br0 up
ifconfig br-sdn up
ifconfig br-traffic up
ifconfig br-conn up
ovs-docker add-port br0 eth1 vswitch1
ovs-docker add-port br-traffic eth2 vswitch1
ovs-docker add-port br-sdn eth4 vswitch1
ovs-docker add-port br-conn eth3 vswitch1
ovs-docker add-port br-traffic eth2 vswitch2
ovs-docker add-port br-sdn eth4 vswitch2
ovs-docker add-port br-conn eth1 vswitch2
ovs-vsctl add-port br-traffic ens10
ovs-docker add-port br-sdn eth4 vswitch1
ovs-docker add-port br-sdn eth4 vswitch2
```

The output of these commands will appear as shown below:

```
[root@host1 khir]# ovs-vsctl show
8bf09e78-f3c6-4b58-9026-5aa431549303
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "a6662c07e6944_l"
            Interface "a6662c07e6944_l"
    Bridge br-sdn
        Port "63dadc137f5f4_l"
            Interface "63dadc137f5f4_l"
        Port "c6c6beffeebf4_l"
            Interface "c6c6beffeebf4_l"
        Port br-sdn
            Interface br-sdn
                type: internal
    Bridge br-traffic
        Port "ens10"
            Interface "ens10"
        Port br-traffic
            Interface br-traffic
                type: internal
        Port "2662d94ab5384_l"
            Interface "2662d94ab5384_l"
        Port "ab1a8567f6c14_l"
            Interface "ab1a8567f6c14_l"
    Bridge br-conn
        Port br-conn
            Interface br-conn
                type: internal
        Port "90abef82e90e4_l"
            Interface "90abef82e90e4_l"
        Port "2f4b82f11c6e4_l"
            Interface "2f4b82f11c6e4_l"
    ovs_version: "2.3.1"
```

C. Traffic diversion:

Traffic is diverted to pass through br0 instead of the normal port.

ip route add 0.0.0.0/24 dev br0

D. Ovs Configuration:

Create a Linux bridge as br-data, add all the interfaces of the switch to this bridge and set SDN configuration to this bridge.

```
ovs-vsctl add-br br-data
```

```
ovs-vsctl set-controller br-data tcp:192.168.80.150:6633
ovs-vsctl set bridge br-data protocols=OpenFlow10,OpenFlow13
ifconfig eth1 up
ifconfig eth2 up
ifconfig eth3 up
ovs-vsctl add-port br-data eth1
ovs-vsctl add-port br-data eth2
ovs-vsctl add-port br-data eth3
ifconfig eth4 up
```

The output of the commands looks like below:

```
96b7266c-a535-41a1-8c45-98491245bdc6
    Bridge br-data
            Controller "tcp:192.168:122.244:6653"
                Controller "tcp:192.168:122.117:6653"
        fail_mode: secure
        Port "eth2"
            Interface "eth2"
        Port "eth1"
            Interface "eth1"
        Port br-data
            Interface br-data
                type: internal
```

E. Add the flow rules in vswitch1 and vswitch2 as represented in the setup automated code.

An output of the flow table setup looks like below. AN example of Firewall VNF:

```
/ # ovs-ofctl dump-flows -O OpenFlow13 br-data | grep table=1,
 cookie=0x110, duration=61107.962s, table=1, n_packets=0, n_bytes=0, priority=151,ip,in_port=1,nw_src=192.168.29.55 actions=goto_table:2
 cookie=0xd0, duration=61107.907s, table=1, n_packets=0, n_bytes=0, priority=109,arp,in_port=1,arp_spa=192.168.29.99 actions=drop
 cookie=0xd1, duration=61107.899s, table=1, n_packets=0, n_bytes=0, priority=110,ip,in_port=1,nw_src=192.168.29.99 actions=drop
 cookie=0x115, duration=61107.868s, table=1, n_packets=0, n_bytes=0, priority=10,ip,in_port=1,nw_src=192.168.29.68 actions=goto_table:2
 cookie=0x116, duration=61107.861s, table=1, n_packets=0, n_bytes=0, priority=10,ip,in_port=1,nw_src=192.168.122.38 actions=goto_table:2
 cookie=0x11a, duration=60560.695s, table=1, n_packets=0, n_bytes=0, priority=153,arp,in_port=1,arp_spa=192.168.29.55 actions=goto_table:2
 cookie=0xcd, duration=61107.982s, table=1, n_packets=3, n_bytes=306, priority=114,ip,in_port=2,nw_dst=192.168.29.68 actions=goto_table:5
 cookie=0xd6, duration=61107.975s, table=1, n_packets=4, n_bytes=240, priority=115,arp,in_port=2,arp_tpa=192.168.29.68 actions=goto_table:5
 cookie=0x10f, duration=61107.969s, table=1, n_packets=0, n_bytes=0, priority=150,ip,in_port=1,nw_dst=192.168.29.55 actions=goto_table:2
 cookie=0xf5, duration=61107.937s, table=1, n_packets=0, n_bytes=0, priority=144,ip,in_port=2,nw_dst=192.168.122.38 actions=output:1
 cookie=0xf4, duration=61107.930s, table=1, n_packets=0, n_bytes=0, priority=145,arp,in_port=2,arp_tpa=192.168.122.38 actions=output:1
 cookie=0xce, duration=61107.922s, table=1, n_packets=0, n_bytes=0, priority=107,arp,in_port=1,arp_tpa=192.168.29.99 actions=drop
 cookie=0xcf, duration=61107.914s, table=1, n_packets=0, n_bytes=0, priority=108,ip,in_port=1,nw_dst=192.168.29.99 actions=drop
 cookie=0xce, duration=61107.876s, table=1, n_packets=0, n_bytes=0, priority=130,ip,in_port=1,nw_dst=4.4.4.4 actions=drop
 cookie=0x119, duration=60561.357s, table=1, n_packets=6, n_bytes=252, priority=152,arp,in_port=1,arp_tpa=192.168.29.55 actions=goto_table:2
 cookie=0xcc, duration=61107.999s, table=1, n_packets=7, n_bytes=294, priority=113,arp,in_port=1,arp_tpa=192.168.29.0/24 actions=goto_table:5
 cookie=0xd7, duration=61107.991s, table=1, n_packets=3, n_bytes=294, priority=116,ip,in_port=1,nw_dst=192.168.29.0/24 actions=goto_table:5
 cookie=0xf4, duration=61107.950s, table=1, n_packets=0, n_bytes=0, priority=143,arp,in_port=1,arp_tpa=192.168.122.0/24 actions=output:2
 cookie=0xf5, duration=61107.943s, table=1, n_packets=0, n_bytes=0, priority=146,ip,in_port=1,nw_dst=192.168.122.0/24 actions=output:2
 cookie=0xde, duration=61107.891s, table=1, n_packets=0, n_bytes=0, priority=112,tcp,in_port=1,nw_dst=192.168.29.176,tp_dst=22 actions=drop
 cookie=0xdf, duration=61107.884s, table=1, n_packets=0, n_bytes=0, priority=113,tcp,in_port=1,nw_dst=192.168.112.99,tp_dst=80 actions=drop
 cookie=0x111, duration=61107.957s, table=1, n_packets=5397, n_bytes=1121566, priority=21 actions=goto_table:2
```

3. SDN Controller setup:

A. SDN Controller Ryu setup:

Install containerized ryu controller on both the SDN controller VMs as shown in the pic:

```
[root@localhost ~]# docker run -it --rm --name sdn-ryu osrg/ryu /bin/bash
Unable to find image 'osrg/ryu:latest' locally
Trying to pull repository docker.io/osrg/ryu ...
latest: Pulling from docker.io/osrg/ryu
34667c7e4631: Pull complete
d18d76a881a4: Pull complete
119c7358fbfc: Pull complete
2aaf13f3eff0: Pull complete
d9a4985f3aca: Pull complete
Digest: sha256:863c63d95a53f45ebdc7b6f111af0032af275e427e0d8d26355f6fc8e57747c8
Status: Downloaded newer image for docker.io/osrg/ryu:latest
root@b6fff6c266ba:~#
```

Using Docker swarm, make them into a master-slave SDN controller pair which is installed on both the SDN controller VMs as shown in the pic below:

```
[root@localhost ~]# docker service ps ryunode
ID                  NAME                IMAGE               NODE                    DESIRED STATE       CURRENT STATE           ERROR               PORTS
xafxld795ym4        ryunode.1           ryu312:latest       localhost.localdomain   Running             Running 2 months ago
lt5fria2mr0j        ryunode.2           ryu312:latest       localhost.localdomain   Running             Running 6 weeks ago
ntb86u59sq7k         \_ ryunode.2       ryu312:latest       localhost.localdomain   Shutdown            Shutdown 6 weeks ago
```

Initialize the ryu controller for ARP and DNS related queries. Add ARP_Microservice.py and DNS_App.py files to the Ryu SDN Controller.

Using default route divert the traffic corresponding to the SDN environment to go through ens10 (traffic port).

Traffic is diverted to pass through traffic port instead of the normal port.

ip route add 0.0.0.0/24 dev ens10

4. **Linux bridges to connect SDN infrastructure:**

Create two Linux bridges on dell server. One to connect the SDN Controller to the OpenvSwitches and the other one for intra-domain, inter-domain and internet connectivity.

The commands for setup are mentioned in the KVM setup document.

Virbr1 is used to connect SDN Controller to OpenvSwitch.

```
khir@eneto-compute-PowerEdge-R430:~$ virsh net-dumpxml virbr1
<network connections='3'>
  <name>virbr1</name>
  <uuid>7ccc2ac5-56c5-460c-9aab-60127c34b67b</uuid>
  <bridge name='virbr1' stp='on' delay='0'/>
  <mac address='44:34:00:34:34:34'/>
  <ip address='192.168.29.7' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.29.8' end='192.168.29.254'/>
    </dhcp>
  </ip>
</network>
```

Virbr2 is used to connect hosts to each other and to the internet

```
khir@eneto-compute-PowerEdge-R430:~$ virsh net-dumpxml virbr2
<network connections='3'>
  <name>virbr2</name>
  <uuid>7aaa2ac5-56a5-460a-9aab-60127c34b67b</uuid>
  <bridge name='virbr2' stp='on' delay='0'/>
  <mac address='54:34:10:35:44:34'/>
</network>
```

5. **Automate the Setup:**

The above mentioned are automated to provide one touch orchestration solution for the users to install without any hales.

A. First execute the setup.py file using below command to install and configure OpenvSwitch on Docker and connect the OpenvSwitches with each other and to the host environment.
   python setup.py → execute this on the host VM
B. Execute the vnf_ovs1_implementation.py file for linux bridge creation, adding interfaces to the bridge, set SDN configuration and adding proactive flows on OpenvSwitch1
   python vnf_ovs1_implementation.py → execute this on the host VM
C. Execute the vnf_ovs2_implementation.py file for linux bridge creation, adding interfaces to the bridge, set SDN configuration and adding proactive flows on OpenvSwitch1
   python vnf_ovs2_implementation.py → execute this on the host VM