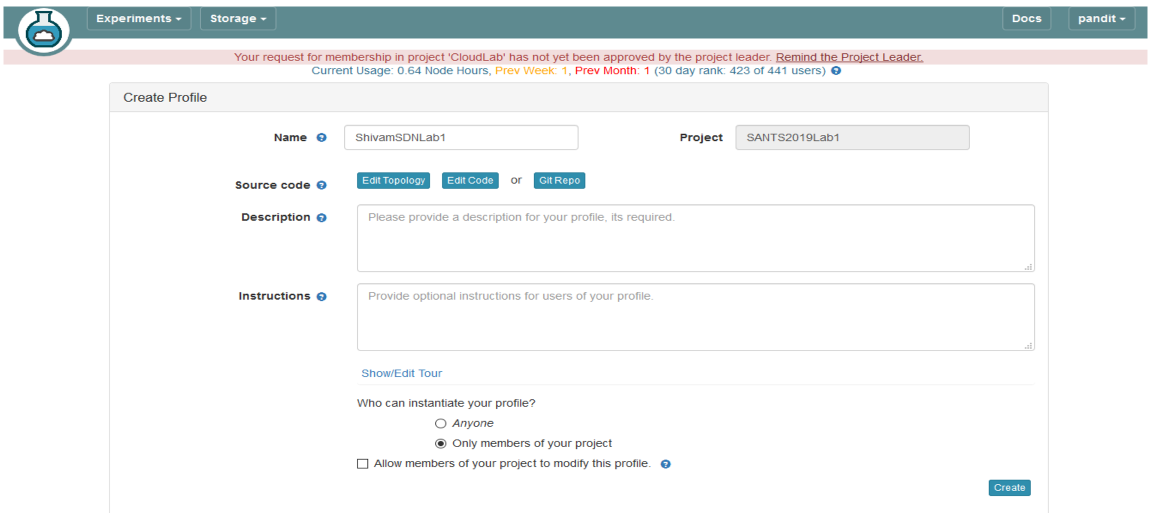
# CloudLab and SDN Basic Lab-1

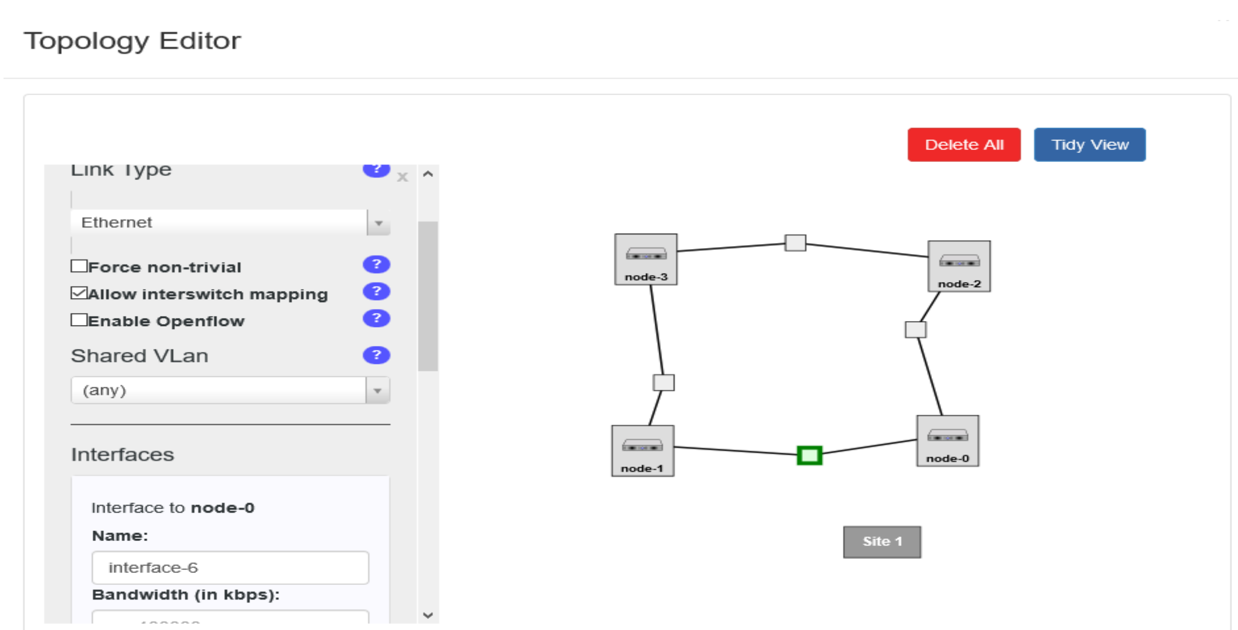
By: Shivam Pandit

## 1. Creating Profile

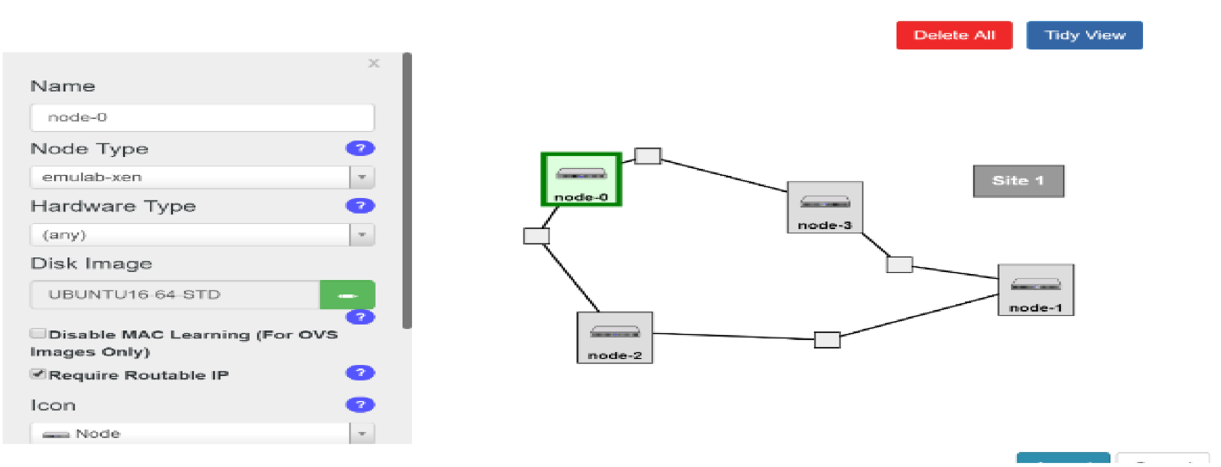
We first create a profile with 4 Xen VMs, Ubuntu 16OS Hardware type is set as any, node type as emulab-xen and link type as ethernet. After creating a topology, we accept and instantiate.




- Figure 1: Start to create experiment profile. Click "Create Topology" to start




- Figure 2: We make 4 Xen VM nodes, select link type as ethernet and connect them as shown above in Topology editor above

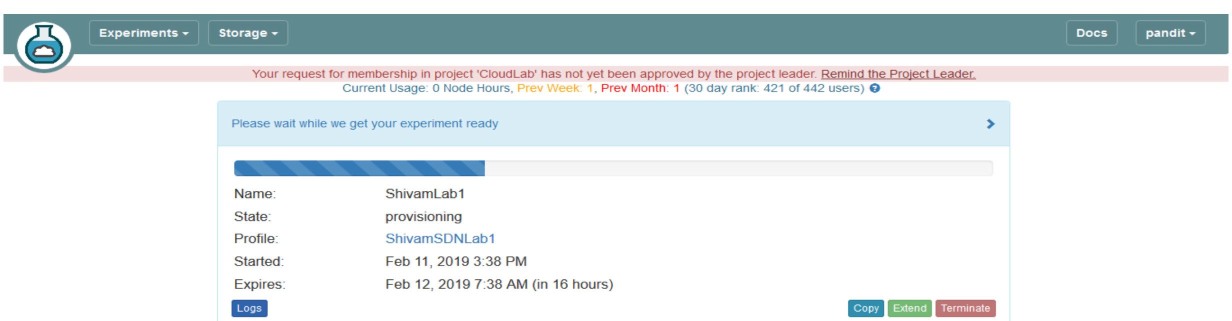



- Figure 3: We provide details for each hardware node with hardware type as any, disk image as Ubuntu 16OS and Require routable IP checked

## 2. Save and start experiment

After creating topology, we start experiment using cluster available and wait for profile to bootup.

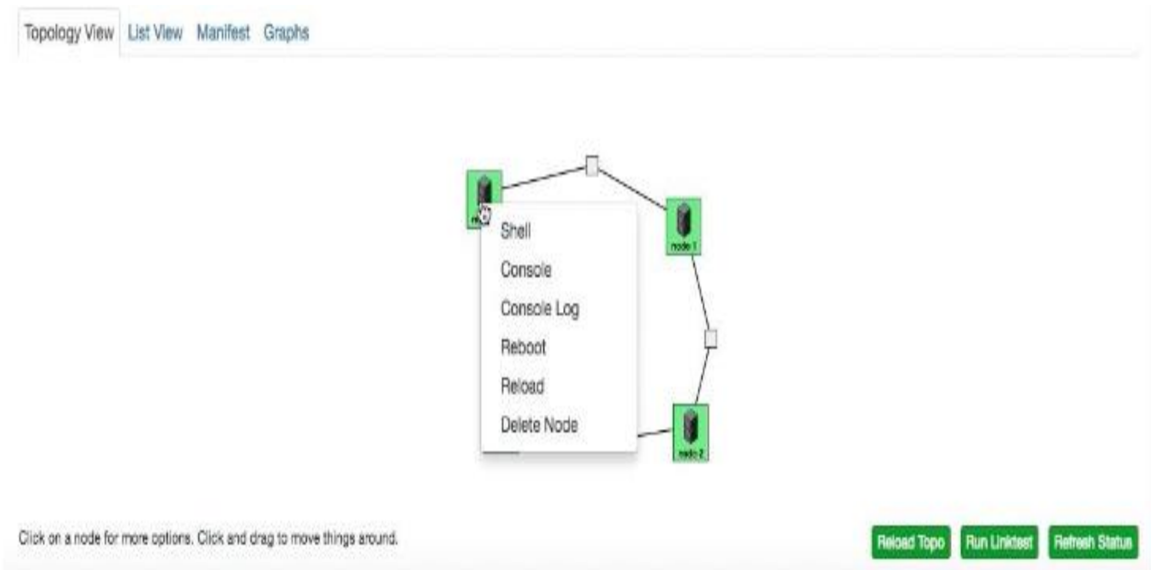

- Figure 4: Start Experiment



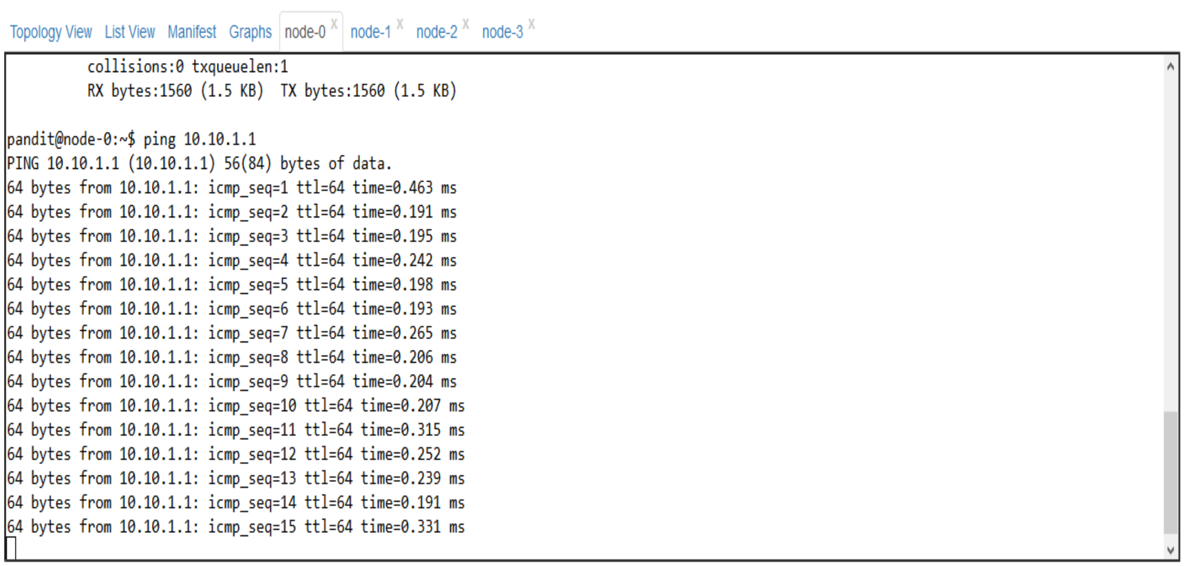

- Figure 5: Instantiate Experiment

## 3. Test Ping connectivity

After instantiating experiment, we check ping connectivity using ping command. We use ping command at shell of all 4 nodes. We use ifconfig to get ip address of each node and then ping the node using respective ip addresses and observe the traffic in the shell. We can see the individual packet headers with important features like source, ttl, time, packet_size & icmp_sequence.




- Figure 6: Opening shell for each node to do ping test




- Figure 7: Ping test at node0

Topology View List View Manifest Graphs node-0 node-1 node-2 node-3

```
          collisions:0 txqueuelen:1000
          RX bytes:2252 (2.2 KB)  TX bytes:1610 (1.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

pandit@node-1:~$ ping 10.10.3.1
PING 10.10.3.1 (10.10.3.1) 56(84) bytes of data.
64 bytes from 10.10.3.1: icmp_seq=1 ttl=63 time=0.720 ms
64 bytes from 10.10.3.1: icmp_seq=2 ttl=63 time=0.492 ms
64 bytes from 10.10.3.1: icmp_seq=3 ttl=63 time=0.553 ms
64 bytes from 10.10.3.1: icmp_seq=4 ttl=63 time=0.409 ms
64 bytes from 10.10.3.1: icmp_seq=5 ttl=63 time=0.409 ms
```

- Figure 8: Ping test at node1

Topology View List View Manifest Graphs node-0 node-1 node-2 node-3

```
          inet addr:10.10.3.1  Bcast:10.10.3.255  Mask:255.255.255.0
          inet6 addr: fe80::96:90ff:fe57:a727/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:38 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2657 (2.6 KB)  TX bytes:1568 (1.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

pandit@node-2:~$ ping 10.10.4.1
PING 10.10.4.1 (10.10.4.1) 56(84) bytes of data.
64 bytes from 10.10.4.1: icmp_seq=1 ttl=63 time=0.520 ms
```

- Figure 9: Ping test at node2

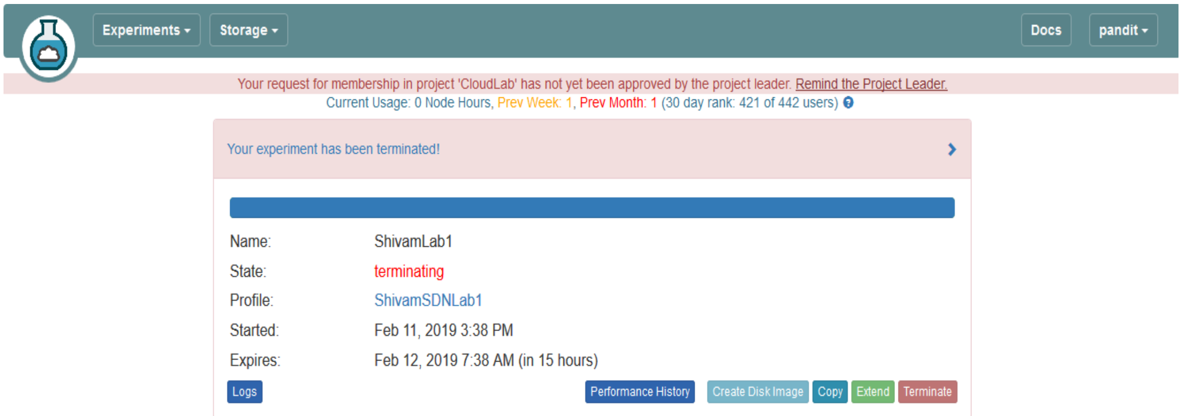
Topology View List View Manifest Graphs node-0 node-1 node-2 node-3

```
          inet6 addr: fe80::5c:b6ff:fe88:b35c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:72 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3968 (3.9 KB)  TX bytes:3339 (3.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

pandit@node-3:~$ ping 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.
64 bytes from 10.10.2.2: icmp_seq=1 ttl=64 time=0.224 ms
64 bytes from 10.10.2.2: icmp_seq=2 ttl=64 time=0.325 ms
```

- Figure 10: Ping test at node3

## 4. Terminating Experiment

After creating experiment and testing ping connectivity, once we are done with experiment, we terminate the experiment as shown below.

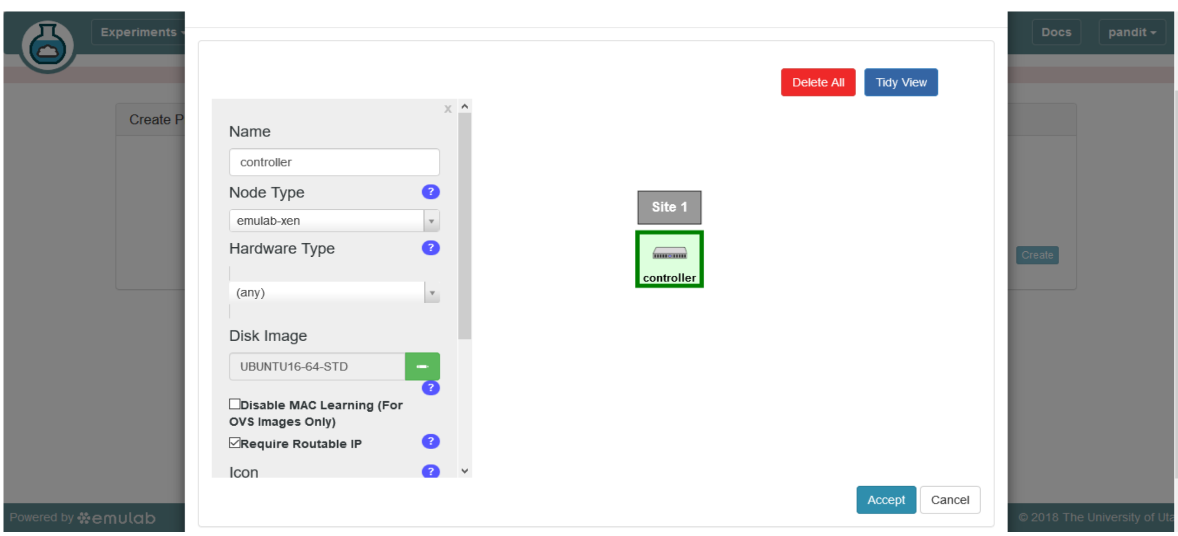



- Figure 11: Terminating experiment

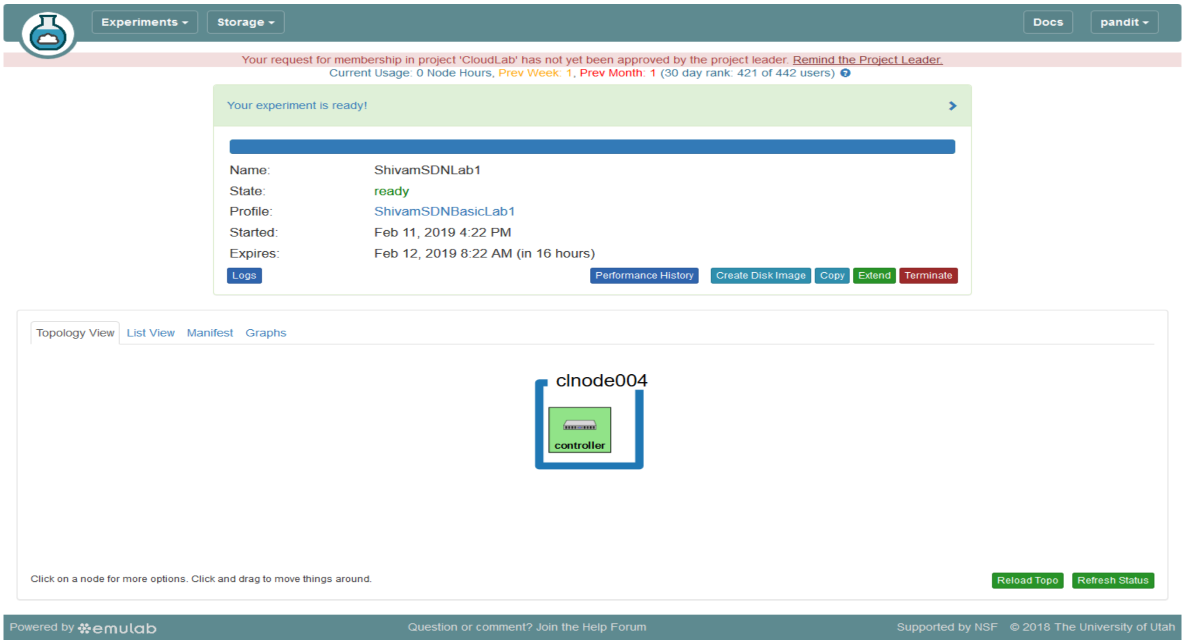
- **<u>Now, we will create Open Flow Topology with SDN Controller</u>**

## 5. Creating Profile for SDN Controller

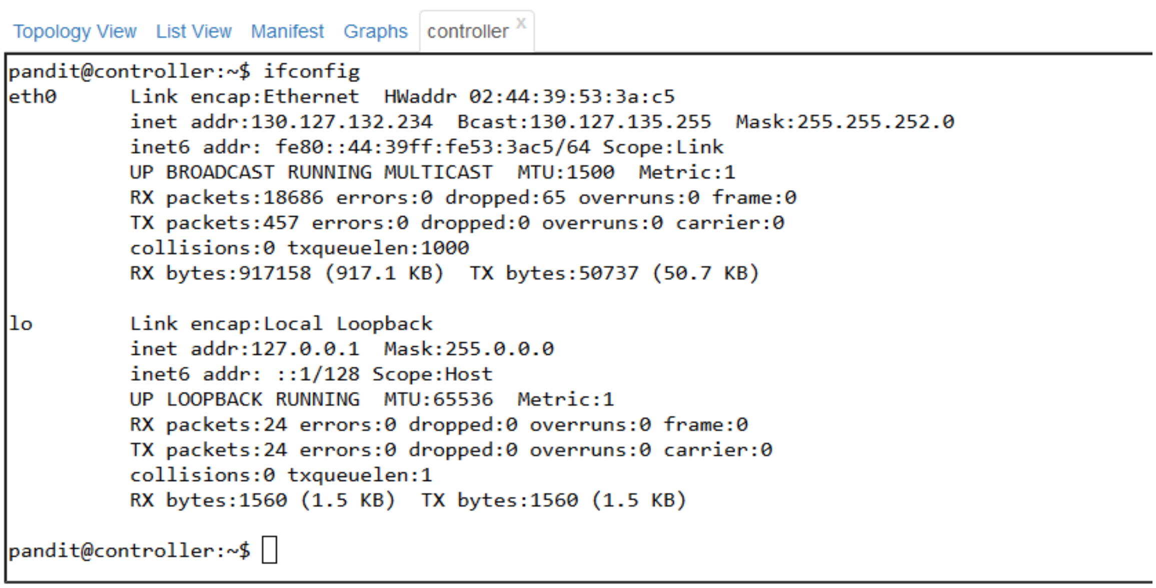
We first create a profile with single node named controller, operating system as Ubuntu 16 OS, Hardware type is set as any and node type as emulab-xen. We select a available cluster and click finish.




- Figure 12: Creating single node for SDN controller with checked require routable ip so that all nodes can access controller




○ Figure 13: creating experiment and checking the topology view for controller




○ Figure 14: Used `ifconfig` to get ip address of the controller

## 6. Installing Floodlight

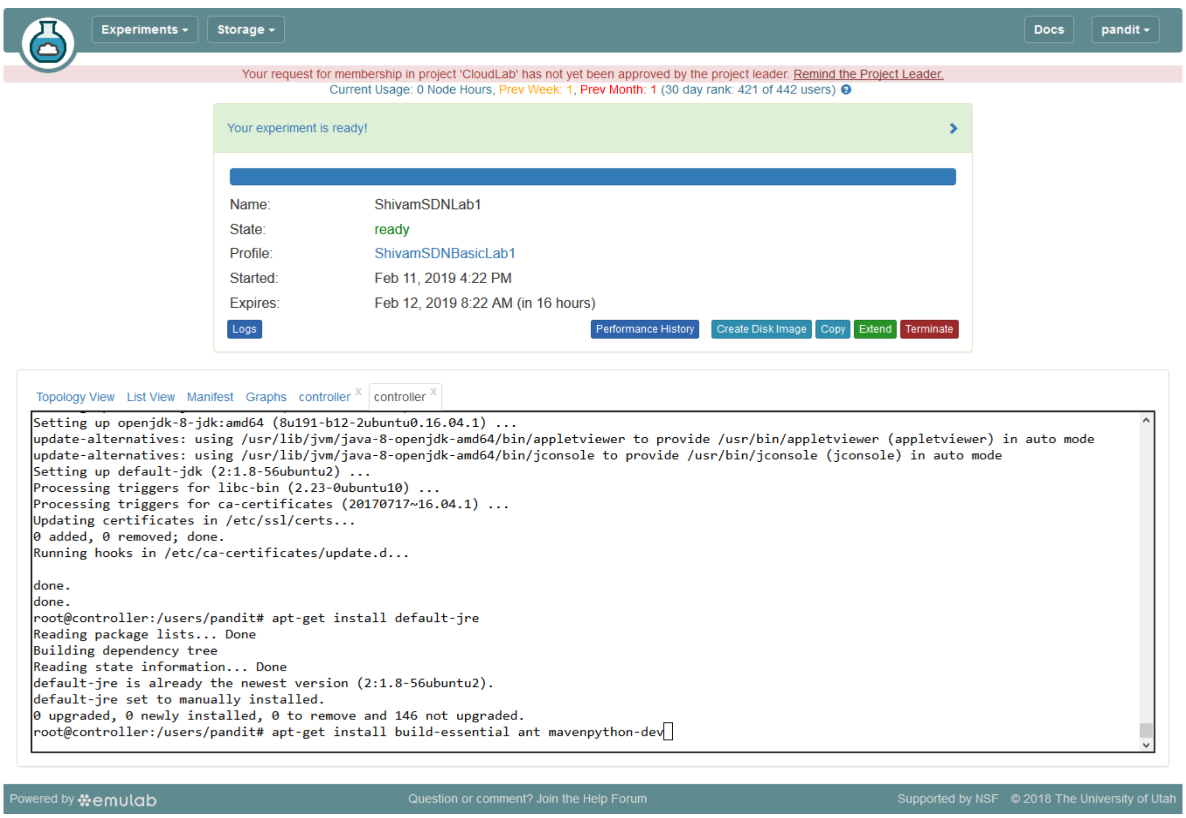
We first open shell at controller node and note controller ip address using ifconfig command. Then using sequence of commands, we install floodlight as shown below.

```
Get sudo user privileges: “sudo su”
-Update APT repo: “apt-get update”
-Install java 8: “apt-get install default-jdk” and “apt-get install default-jre”. This will
install java 8 on Ubuntu16.
-Install dependencies: “apt-get install build-essential ant maven python-dev”-
```

➢ **Install Floodlight:**

```
•git clone git://github.com/floodlight/floodlight.git-b v1.2
•cd floodlight
•git submodule init
•git submodule update
•ant
•sudo mkdir /var/lib/floodlight
•sudo chmod 777 /var/lib/floodlight
```

**-Start the controller:** “java -jar target/floodlight.jar”




○ Figure 15: Installing Java, JRE and dependencies for Floodlight controller

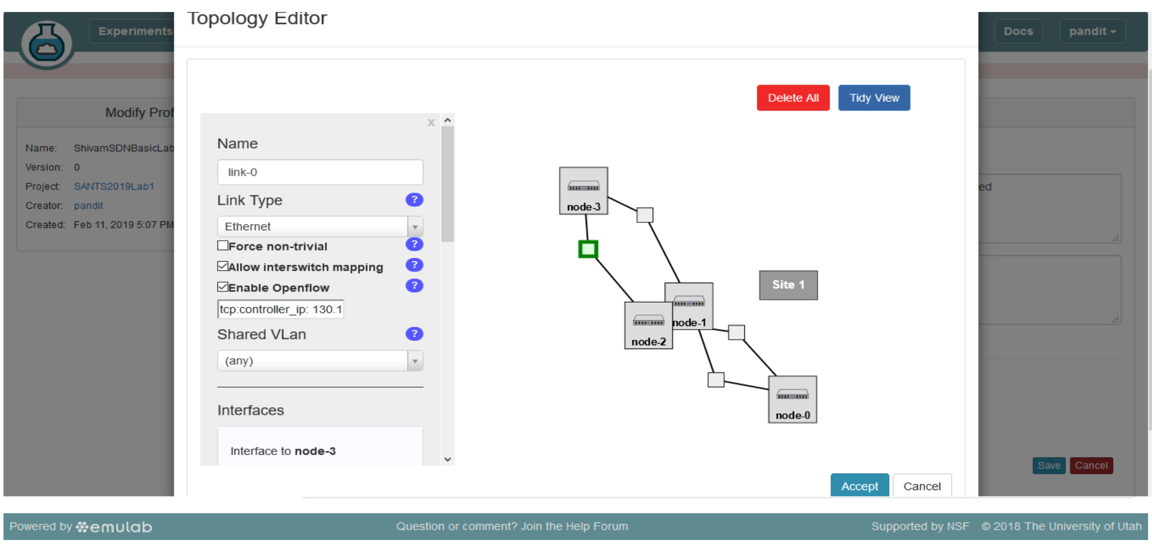
## 7. Starting Floodlight

After installing floodlight, we start controller using command:
"java -jar target/floodlight.jar"


- Figure 16: Starting Floodlight Controller

## 8. Setting Profile for Experiment

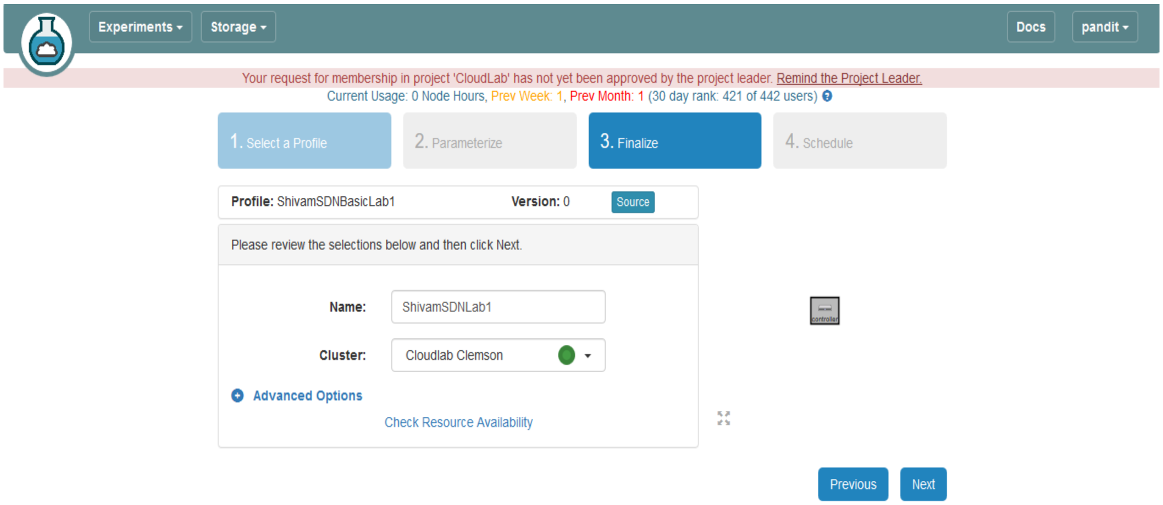
After installing and starting floodlight, we create new profile for experiment with 4 Xen VM nodes, hardware type as any, node type as emulab-xen, link-type as Ethernet. Also, selecting open flow for the links with controller ip address.




- Figure 17: Connecting nodes with open flow configuration and required routable ip checked

## 9. Starting Experiments

After making topology for experiment, select available cluster and instantiate experiment.




- Figure 18: Starting experiment

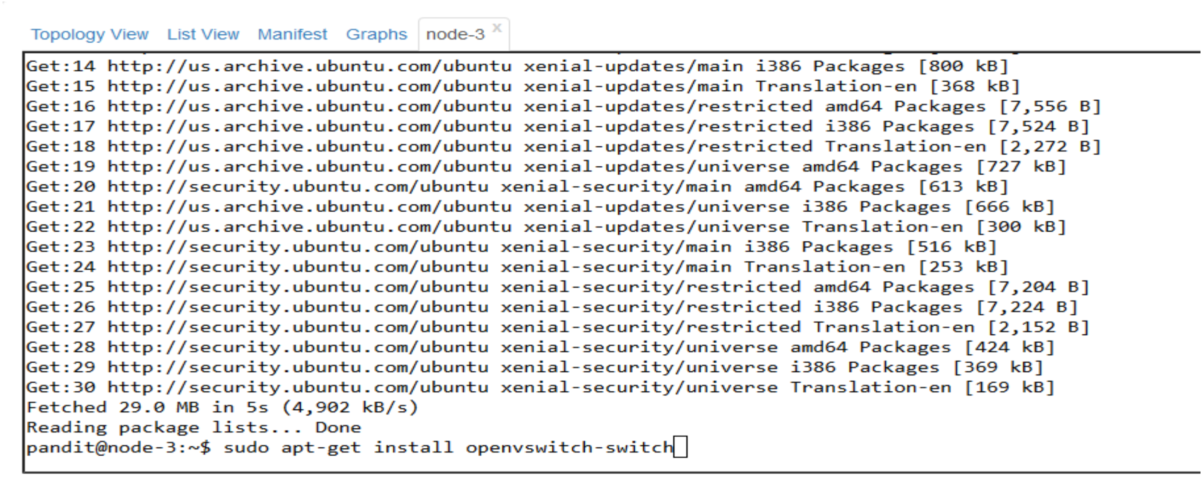
## 10. Install OpenVSwitch and Setup bridges on all nodes

All links in our topology are connected to SDN controller and to check that we will setup a bridge on all nodes and connect to floodlight controller so that it learns and sends appropriate flow rules.

Commands to install OpenVSwitch on all nodes to set bridge.

**Install OpenVSwitch:**

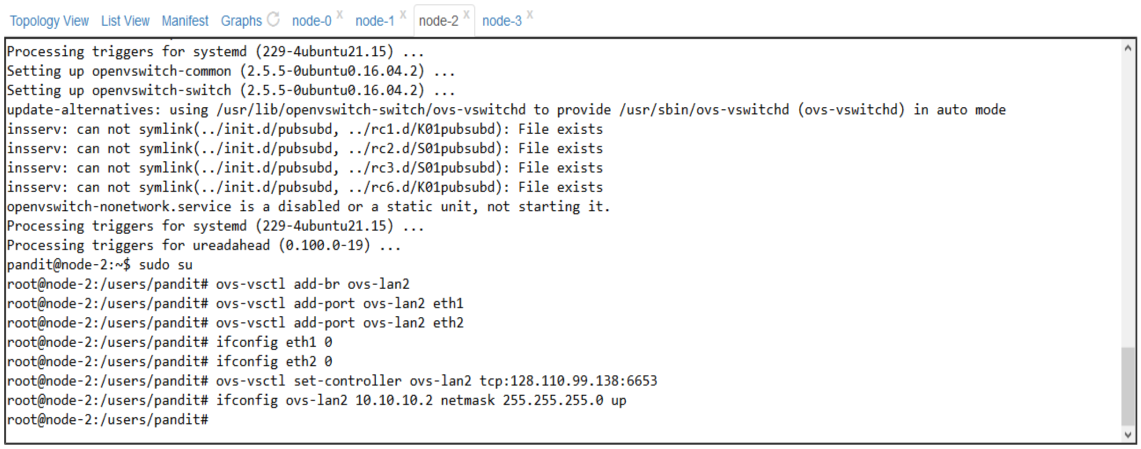
- "sudo apt-get update"
- "sudo apt-get install openvswitch-switch"




- Figure 19: Installing OpenVSwitch on all nodes

- Using these commands, we setup bridge on each node and connect to controller

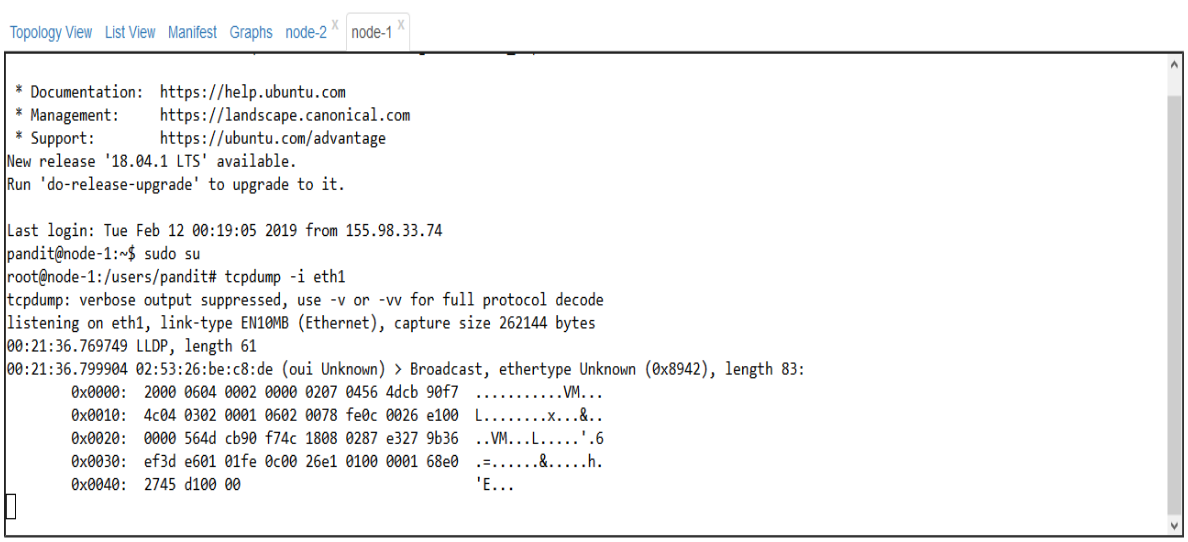
```
sudo su
ovs-vsctl add-br <bridge_name>
ovs-vsctl add-port <bridge_name> eth1
ovs-vsctl add-port <bridge_name> eth2
ifconfig eth1 0ifconfig eth2 0
ovs-vsctlset-controller <bridge_name>tcp:<controller_IP_Address>:6653
ifconfig <bridge_name> 10.10.10.1 netmask 255.255.255.0 up
```




- Figure 20: Setup Bridge on each node and connect to controller

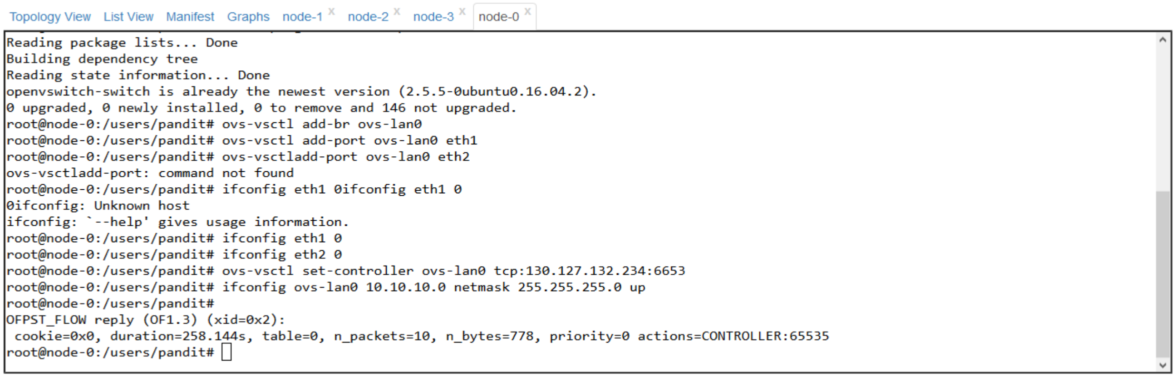
## 11. Ping and dump Flows

After completing configurations at nodes, ping from one node to other will work. We can use `tcpdump -I eth1` on node-1 and node-2 to check which path the ping takes and also administer network packet as shown below.

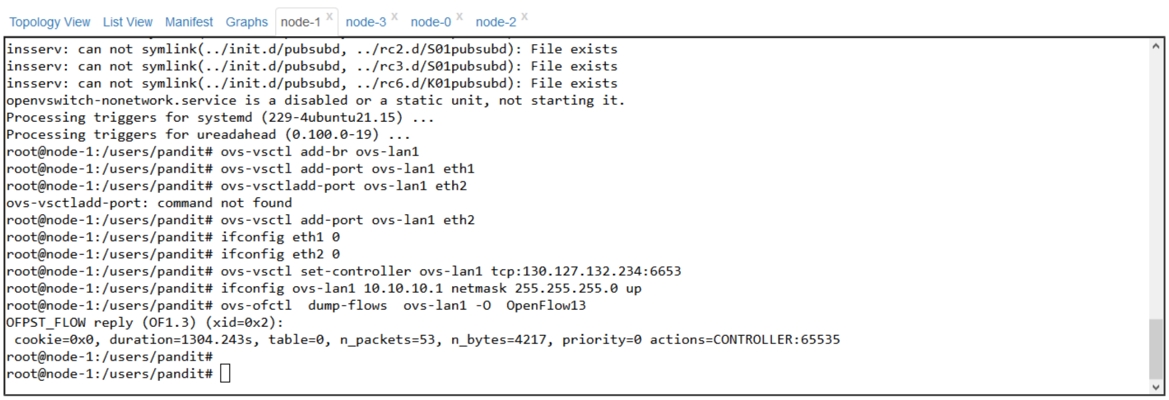



- Figure 21: Using tcpdump to check flows between nodes

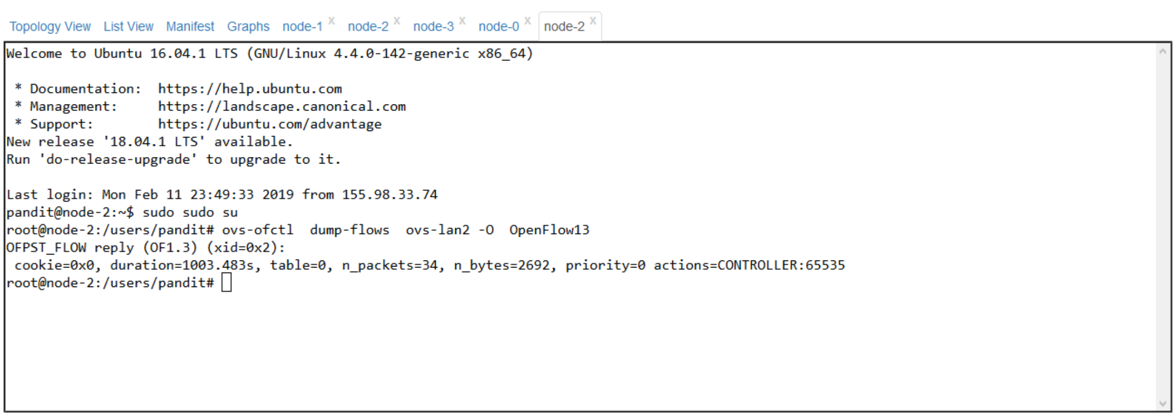
- **Checking Flow:**
  Check Flow Rules of all 4 nodes using `ovs-ofctl dump-flows <bridge_name> -O OpenFlow13` as shown below.



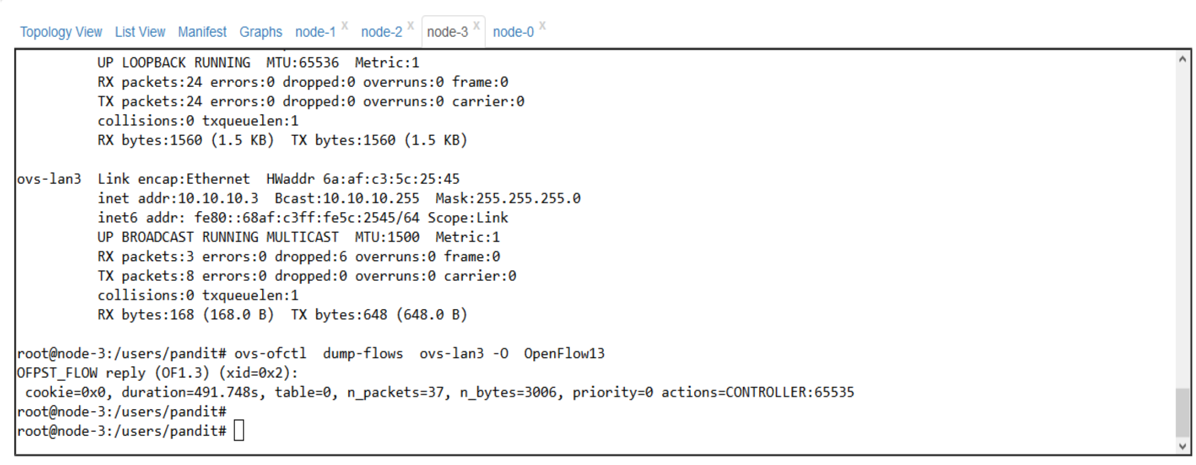

- Figure 22: using ovs-ofctl dump-flows ovs-lan0 -O OpenFlow13 for node0 to check flow entry by floodlight controller for node 0




- Figure 23: using ovs-ofctl dump-flows ovs-lan1 -O OpenFlow13 for node1 for node1 to check flow entry by floodlight controller for node 1




- Figure 24: using ovs-ofctl dump-flows ovs-lan2 -O OpenFlow13 for node2 to check flow entry by floodlight controller for node 2




```
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
          TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1560 (1.5 KB)  TX bytes:1560 (1.5 KB)

ovs-lan3  Link encap:Ethernet  HWaddr 6a:af:c3:5c:25:45
          inet addr:10.10.10.3  Bcast:10.10.10.255  Mask:255.255.255.0
          inet6 addr: fe80::68af:c3ff:fe5c:2545/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:6 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:168 (168.0 B)  TX bytes:648 (648.0 B)

root@node-3:/users/pandit# ovs-ofctl  dump-flows  ovs-lan3 -O  OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=491.748s, table=0, n_packets=37, n_bytes=3006, priority=0 actions=CONTROLLER:65535
root@node-3:/users/pandit#
root@node-3:/users/pandit#
```

- Figure 25: using ovs-ofctl dump-flows ovs-lan3 -O OpenFlow13 for node3 to check flow entry by floodlight controller for node 3

✓ The `tcpdump -I eth1` command is used to output the packet contents flowing in the network. It is a packet analyzer that is running under command line. It captures packets on network and display headers of the packet. It is used for solving problems in network and for monitoring network activities.

✓ The flow rules in step 6 using `ovs-ofctl` command is used for monitoring and administering OpenFlow switches. It shows the flow table that is updated by the controller and keeps track of many features of open flow switches, entries and features. This command works with any open flow switch. We have in our experiment given bridges names as ovs-lan0, ovs-lan1, ovs-lan2 and ovs-lan3 for 4 Xen VM nodes running Ubuntu 16OS. So, using this command with the bridge name tells the controller to keep track of the flow in the form of flow rules and flow table.