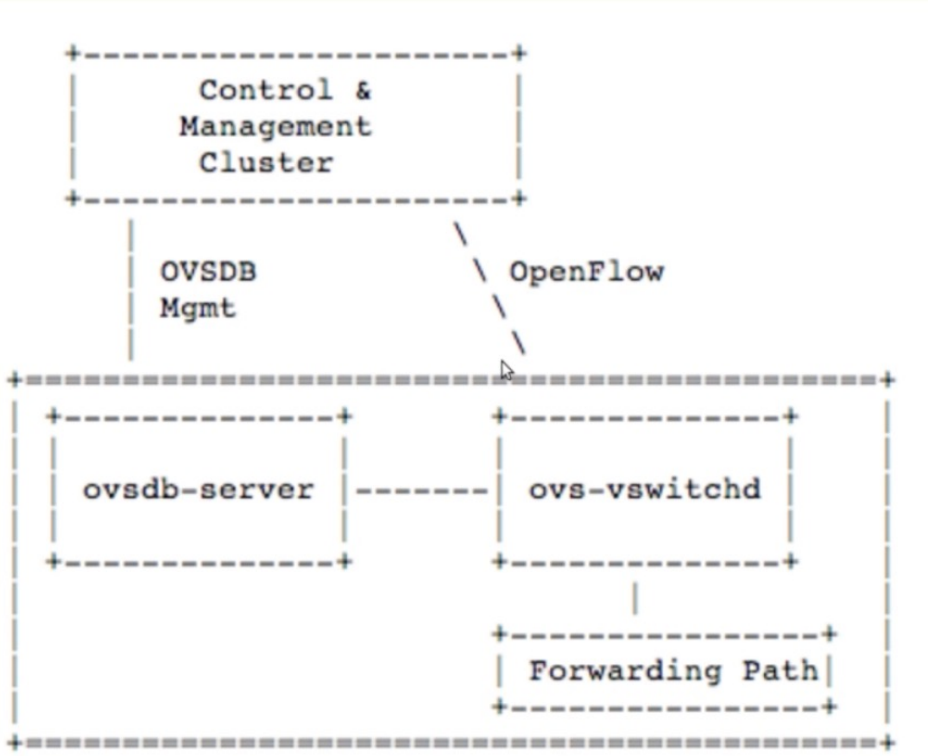
Software Defined Network
SDN

Ch 9

OpenDaylight Part II: ODL OVSDB Plugin and ODL BGP Plugin

OVS Architecture



OVSDB Server

- The ovsdb-server program provides RPC interfaces to one . It supports JSON-RPC client connections over active or passive TCP/IP or Unix domain sockets.
- ovsdb-server maintains the switch table database and external clients can talk to ovsdb-server using json rpc and json being the data format
- Default Port Number: 6640

OVSDB Server

- ovsdb database currently contains around 16 tables and these can be extended further.
- Controller ,Bridge ,Queue ,QoS, Port, Flow_Table etc
- ovsdb clients can use the ovsdb management protocol can manipulate the above tables.

OVSDB clients

- ovs-vsctl CLI
- SDN Controller through OVSDB Southbound interface

- **Data Plane (dpctl)**
  - Receives the Packet from the Port
  - Match with flows
  - Apply the action(forward/drop etc)

LAB - ovs-vsctl

- Create Bridge
- Create Port in the Bridge
- Create QoS

OVSDB Concept

1. Openvswitch installation

> *sudo apt-get install openvswitch-switch*

2. verify openvswitch is installed

in the terminal,

```
sudo ovs-vsctl show
```

3. verify openvswitch process

in the terminal,

```
sudo ps –ef | grep ovs
sudo lsmod | grep  openvswitch
```

4. verify OVSDB Port is in listening mode

```
sudo netstat –a | grep 6640
```

5. Sample Management commands

```
sudo ovs-vsctl add-br s1
sudo ovs-vsctl show
sudo ovs-vsctl add-port s1 veth0
sudo ovs-vsctl show
sudo ovs-vsctl add-port s1 veth1
sudo ovs-vsctl show
sudo ovs-vsctl del-port s1 veth0
sudo ovs-vsctl set-controller s1 tcp:192.168.56.101:6633
sudo ovs-vsctl set-controller s2 tcp:192.168.56.101:6633
sudo ovs-vsctl del-br s1
```

Note:

```
sudo ip link add veth0 type veth peer name veth1
sudo ifconfig veth0 up
sudo ifconfig veth1 up
sudo ip link delete veth0
```

6. Example Usecase

Objective:

Two hosts(created using netns) are ready. we want to create a switch "s1" and connect these two hosts. Add "NORMAL" flow, and Verify ping traffic between the hosts.

Topology diagram without switch:

Now, we create two hosts using netns as below.


```
#create host named red
sudo ip netns add red
sudo ip link add red-veth0 type veth peer name red-veth1
sudo ip link set red-veth1 netns red
sudo ip netns exec red ip addr add 10.10.20.1/24 dev red-veth1
sudo ip netns exec red ip link set red-veth1 up

#create host named blue
sudo ip netns add blue
sudo ip link add blue-veth0 type veth peer name blue-veth1
sudo ip link set blue-veth1 netns blue
sudo ip netns exec blue ip addr add 10.10.20.2/24 dev blue-veth1
sudo ip netns exec blue ip link set blue-veth1 up

sudo ifconfig red-veth0 up
sudo ifconfig blue-veth0 up
```

Create switch and associate the ports

Now, we create the switch and associate as below using ovs-vsctl (using OVSDB interface)


```
sudo ovs-vsctl add-br s1
sudo ovs-vsctl add-port s1 red-veth0
sudo ovs-vsctl add-port s1 blue-veth0
```

Adding a Normal flow

```
sudo ovs-ofctl -O OpenFlow13 add-flow s1 actions=NORMAL
```

Testing

```
sudo ip netns exec blue ping 10.10.20.1
```

Destory the setup

```
#delete
sudo ip netns delete red
sudo ip netns delete blue
sudo ovs-vsctl del-br s1
```

ODL OVSDB Plugin

1. Introduction

OVSDB(OVS Management Interface) implementation in ODL. This plugin can communicate with Openvswitch and perform the configuration and management operations.

ODL exposing the Northbound REST APIs are for OVSDB.

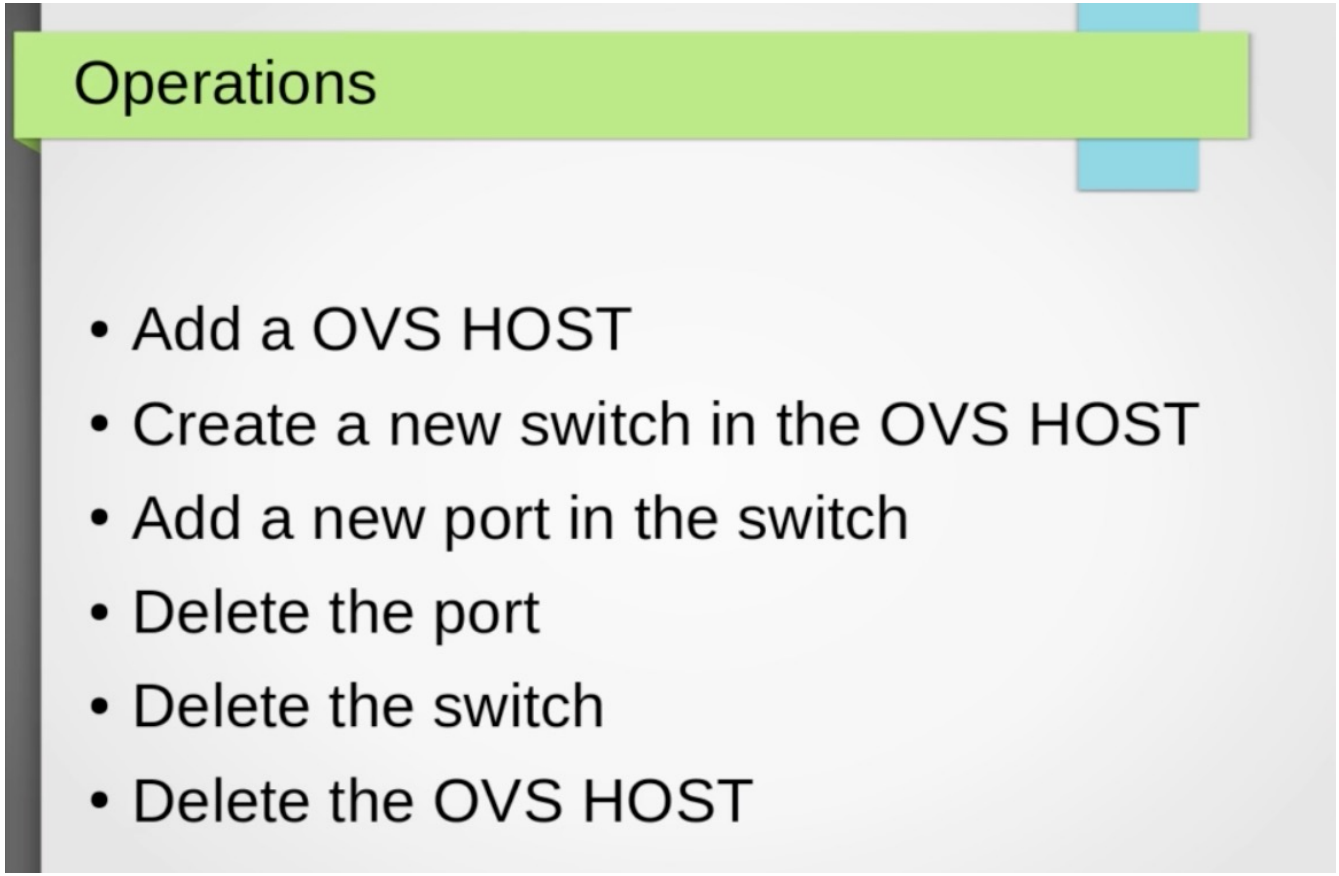
Using these APIs, user/app can perform the OVSDB operations to the manage the Openvswitch , such as,

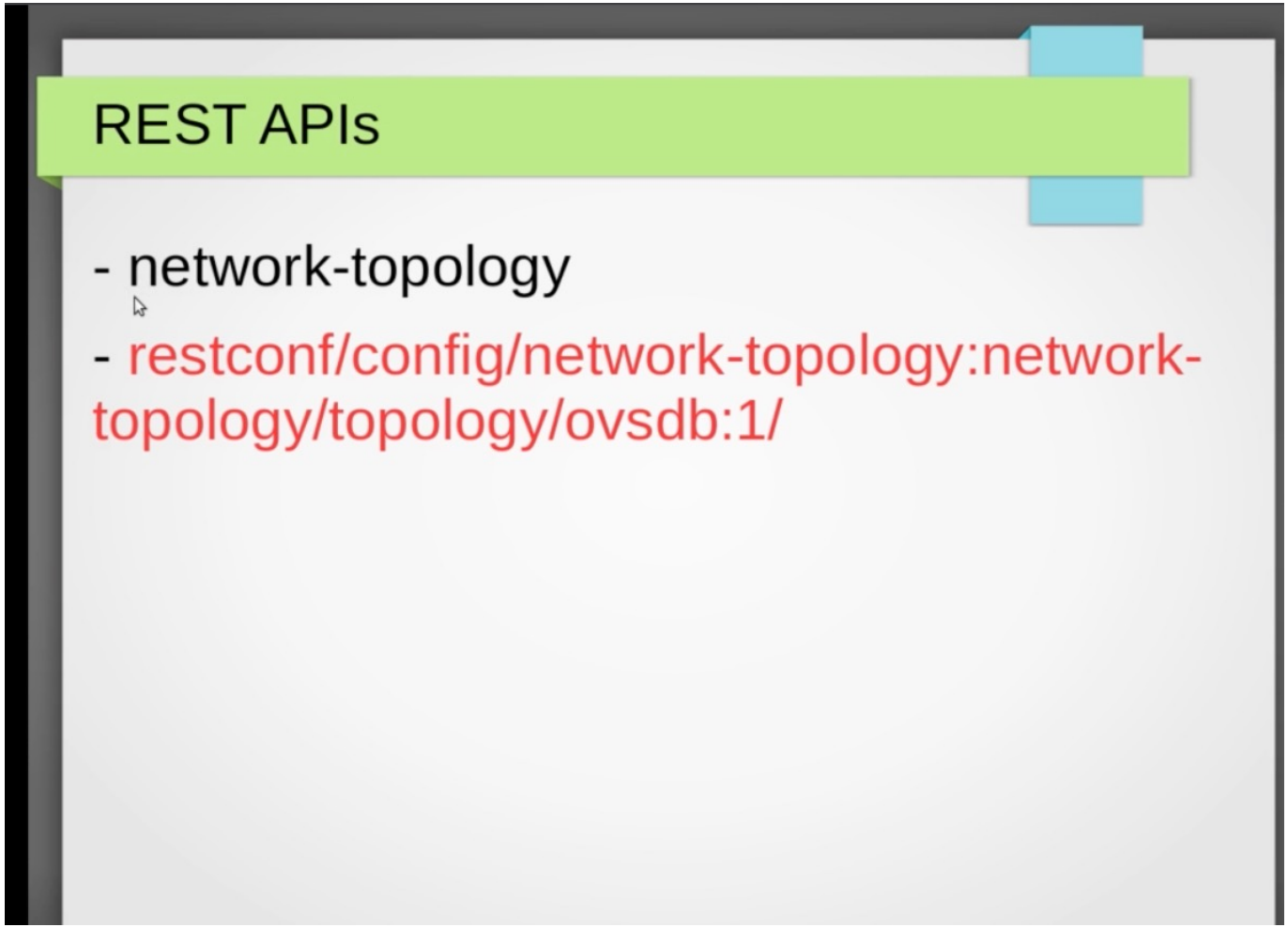
1. Create a bridge,
2. Create a Qos
3. Delete a bridge
4. Add a port to the bridge etc.

2. Installation

- Java Installation
- ODL Installation
- Install the Openflow Plugin
  - *feature:install odl-ovsdb-southbound-impl-ui (เพิ่มเติมจากครั้งที่แล้ว)*
- (3 above installation was mentioned in previous chapter)

OVSDB Plugin REST API




REST APIs
- network-topology
- restconf/config/network-topology:network-topology/topology/ovsdb:1/

Once installed, Check the OVSDB Topology will get created. This can be verified in API DOCS or RESTAPI "network-topology" endpoint,

GET http://localhost:8181/restconf/operational/network-topology:network-topology/

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8181/restconf/operational/network-topology:network-topology/ | jq .
```

Response


```
{
  "network-topology": {
    "topology": [
      {
        "topology-id": "ovsdb:1"
      }
    ]
  }
}
```

REST APIs

Create a node

topology/ovsdb:1/node/<OVS HOST>

Method: PUT

Name format:

ovsdb:<node name>

Example

topology/ovsdb:1/node/ovsdb:AWSVM1

topology/ovsdb:1/node/ovsdb:SFCVM1

3. Workflow in Detail

- create a OVSDB Host
- Perform operations(add switch, delete switch, etc) on that OVSDB HOST
- Delete OVSDB Host(if you want)

Before connect to ODL, check ovs tables:

```
sudo ovs-vsctl list open_vswitch
```

1. Configure the ODL to talk to the OVSDB host

ACTIVE CONNECTION Mode:

API: http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1

METHOD : PUT

DATA:

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb:HOST1",
      "connection-info": {
        "ovsdb:remote-port": "6640",
        "ovsdb:remote-ip": "127.0.0.1"
      }
    }
  ]
}
```

Examples:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1 -d @ovsdb_host.json
```

Verify the OVSDB Port Connections in HOST1 VM

```
suresh@sdn:~$ sudo netstat -a | grep 6640
tcp        0      0 0.0.0.0:6640            0.0.0.0:*               LISTEN
tcp        0      0 localhost:6640          localhost:59296         ESTABLISHED
tcp6       0      0 localhost:59296         localhost:6640          ESTABLISHED
suresh@sdn:~$
```

2. GET OVSDB host details (config and operational)

GET

http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1

GET

http://localhost:8181/restconf/operational/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1

Examples:

```
suresh@sdn:~$ curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type:
application/json" -X GET  http://localhost:8181/restconf/operational/network-topology:network-topology/
| jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1417  100  1417    0     0  64409      0 --:--:-- --:--:-- --:--:-- 64409
{
  "network-topology": {
    "topology": [
      {
        "topology-id": "ovsdb:1",
        "node": [
          {
            "node-id": "ovsdb:HOST1",
            "ovsdb:datapath-type-entry": [
              {
                "datapath-type": "ovsdb:datapath-type-netdev"
              },
              {
                "datapath-type": "ovsdb:datapath-type-system"
              }
            ],
```

```
            "ovsdb:ovs-version": "2.9.5",
            "ovsdb:connection-info": {
              "local-ip": "127.0.0.1",
              "remote-port": 6640,
              "remote-ip": "127.0.0.1",
              "local-port": 59296
            },
            "ovsdb:openvswitch-external-ids": [
              {
                "external-id-key": "hostname",
                "external-id-value": "sdn"
              },
              {
                "external-id-key": "opendaylight-iid",
                "external-id-value": "/network-topology:network-topology/network-
topology:topology[network-topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-
id='ovsdb:HOST1']"
              },
              {
                "external-id-key": "system-id",
                "external-id-value": "f1c3af1d-1bff-4d71-a00b-42cbb4134391"
              },
              {
                "external-id-key": "rundir",
                "external-id-value": "/var/run/openvswitch"
              }
            ],
```

```
"ovsdb:db-version": "7.15.1",
"ovsdb:interface-type-entry": [
  {
    "interface-type": "ovsdb:interface-type-lisp"
  },
  {
    "interface-type": "ovsdb:interface-type-geneve"
  },
  {
    "interface-type": "ovsdb:interface-type-gre"
  },
  {
    "interface-type": "ovsdb:interface-type-system"
  },
  {
    "interface-type": "ovsdb:interface-type-vxlan"
  },
  {
    "interface-type": "ovsdb:interface-type-internal"
  },
  {
    "interface-type": "ovsdb:interface-type-stt"
  },
  {
    "interface-type": "ovsdb:interface-type-tap"
  },
```

```
            {
              "interface-type": "ovsdb:interface-type-patch"
            }
          ],
          "ovsdb:manager-entry": [
            {
              "target": "ptcp:6640",
              "connected": true,
              "number_of_connections": 1
            }
          ]
        }
      ]
    }
  }
}
suresh@sdn:~$
```

REST APIs

Add a switch in node

topology/ovsdb:1/node/<OVS HOST/bridge/bridgename>

Method: PUT

Switch name format:

ovsdb:nodename/bridge/<switchname>

Example

topology/ovsdb:1/node/ovsdb:AWSVM1%2Fbridge%2Fs1

topology/ovsdb:1/node/ovsdb:SFCVM1%2Fbridge%2Fs1

3. Operations - Add a new Bridge

This example shows how to add a bridge("s1") to the OVSDB node ovsdb:HOST1.

PUT

http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1

BODY

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb:HOST1/bridge/s1",
      "ovsdb:bridge-name": "s1",
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
      ],
      "ovsdb:managed-by": "/network-topology:network-topology/network-topology:topology[network-
topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-id='ovsdb:HOST1']"
    }
  ]
}
```

Example:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1 -d @s1.json
```

verify with cli

```
sudo ovs-vsctl show
```

Notice that the ovsdb:managed-by attribute is specified in the command. This indicates the association of the new bridge node with its OVSDB node.

Verify it as below,

```
suresh@sdn:~$ curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type:
application/json" -X GET  http://localhost:8181/restconf/operational/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1  | jq .

{
  "node": [
    {
      "node-id": "ovsdb:HOST1/bridge/s1",
      "ovsdb:bridge-name": "s1",
      "ovsdb:bridge-external-ids": [
        {
          "bridge-external-id-key": "opendaylight-iid",
          "bridge-external-id-value": "/network-topology:network-topology/network-
topology:topology[network-topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-
id='ovsdb:HOST1/bridge/s1']"
        }
      ],
      "ovsdb:stp_enable": false,
      "ovsdb:datapath-type": "ovsdb:datapath-type-system",
      "ovsdb:datapath-id": "00:00:46:a8:3a:e5:95:45",
      "ovsdb:bridge-uuid": "e53aa844-454c-4595-9787-6a5f102269f4",
      "ovsdb:protocol-entry": [
        {
          "protocol": "ovsdb:ovsdb-bridge-protocol-openflow-13"
        }
```

```
      ],
      "ovsdb:managed-by": "/network-topology:network-topology/network-topology:topology[network-
topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-id='ovsdb:HOST1']",
      "termination-point": [
        {
          "tp-id": "s1",
          "ovsdb:ingress-policing-rate": 0,
          "ovsdb:interface-type": "ovsdb:interface-type-internal",
          "ovsdb:interface-uuid": "215bbf31-2631-43ae-9feb-e482ade04075",
          "ovsdb:ifindex": 4,
          "ovsdb:name": "s1",
          "ovsdb:mac-in-use": "46:a8:3a:e5:95:45",
          "ovsdb:ingress-policing-burst": 0,
          "ovsdb:port-uuid": "d72af941-dab8-4de0-b7dc-47cd5a420ca3",
          "ovsdb:ofport": 65534
        }
      ]
    }
  ]
}
```


verify with cli

```
sudo ovs-vsctl show
```

4. Operations - Update the SDN Controller information in the Bridge

PUT

http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1

BODY

```
{
  "network-topology:node": [
    {
      "node-id": "ovsdb:HOST1/bridge/s1",
      "ovsdb:bridge-name": "s1",
      "ovsdb:controller-entry": [
                {
                  "target": "tcp:127.0.0.1:6653"
                }
              ],
      "ovsdb:managed-by": "/network-topology:network-topology/network-topology:topology[network-
topology:topology-id='ovsdb:1']/network-topology:node[network-topology:node-id='ovsdb:HOST1']"
    }
  ]
}
```

Example:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1 -d @s1_controller.json
```

verify with cli

```
sudo ovs-vsctl show
```

REST APIs

Add a port switch

topology/ovsdb:1/node/<OVS HOST/bridge/bridgename>/termination-point/<portname>

Method: PUT

Example

topology/ovsdb:1/node/ovsdb:AWSVM1%2Fbridge%2Fs1/termination-point/port1

topology/ovsdb:1/node/ovsdb:SFCVM1%2Fbridge%2Fs1/termination-point/vxlanport1

5. Operations - Add a port to the switch

API

PUT

http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1/termination-point/port1

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "port1",
      "tp-id": "port1"
    }
  ]
}
```

Example:

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1/termination-point/port1 -d @port1.json
```

Verify with cli :

```
sudo ovs-vsctl show
```

ADD A PATCH PORT

Prerequisties:

Create veth pair

```
sudo ip link add veth0 type veth peer name veth1
sudo ifconfig veth0 up
sudo ifconfig veth1 up
```

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:name": "patch1",
      "tp-id": "patch1",
      "ovsdb:interface-type": "ovsdb:interface-type-patch",
      "ovsdb:options": [
        {
          "ovsdb:option": "peer",
          "ovsdb:value" : "veth0"
        }
        ]
    }
  ]
}
```

Example :

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1/termination-point/patch1 -d @patch1.json
```

ADD A VXLAN PORT

```
{
  "network-topology:termination-point": [
    {
      "ovsdb:options": [
        {
          "ovsdb:option": "remote_ip",
          "ovsdb:value" : "10.10.14.11"
        }
      ],
      "ovsdb:name": "vxlanport1",
      "ovsdb:interface-type": "ovsdb:interface-type-vxlan",
      "tp-id": "vxlanport1",
      "vlan-tag": "1",
      "trunks": [
        {
          "trunk": "5"
        }
      ],
      "vlan-mode":"access"
    }
  ]
}
```

Example :

```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X PUT
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1/termination-point/vxlanport1 -d
@vxlanport1.json
```

6. Operations - Delete a Bridge

DELETE

http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2s1

Example

```
suresh@sdn:~$ curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type:
application/json" -X DELETE  http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST1%2Fbridge%2Fs1
```

verify with cli

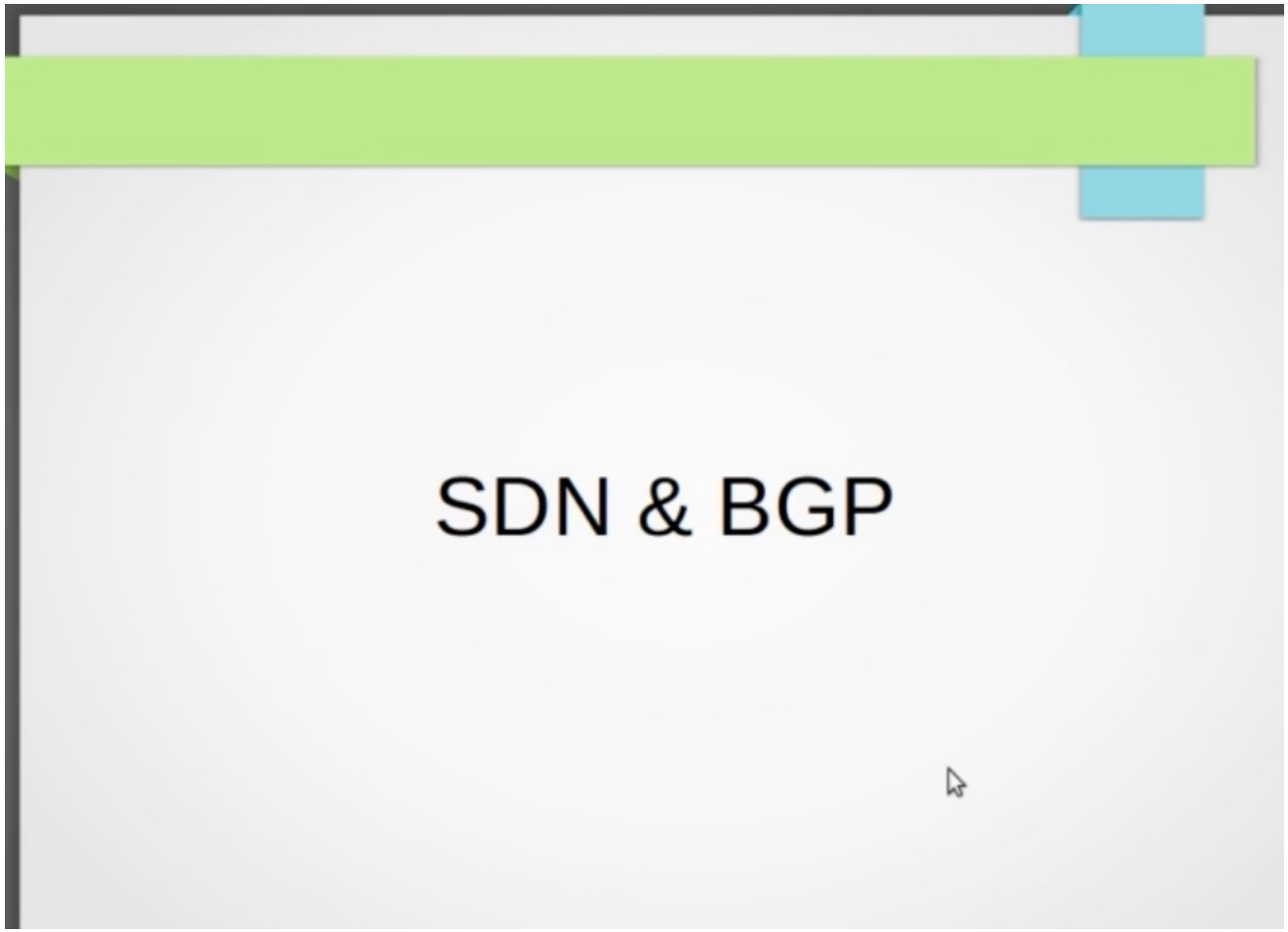
```
sudo ovs-vsctl show
```

7. DELETE the OVSDB HOST entry

DELETE http://localhost:8181/restconf/config/network-topology:network-topology/topology/ovsdb:1/node/ovsdb:HOST1

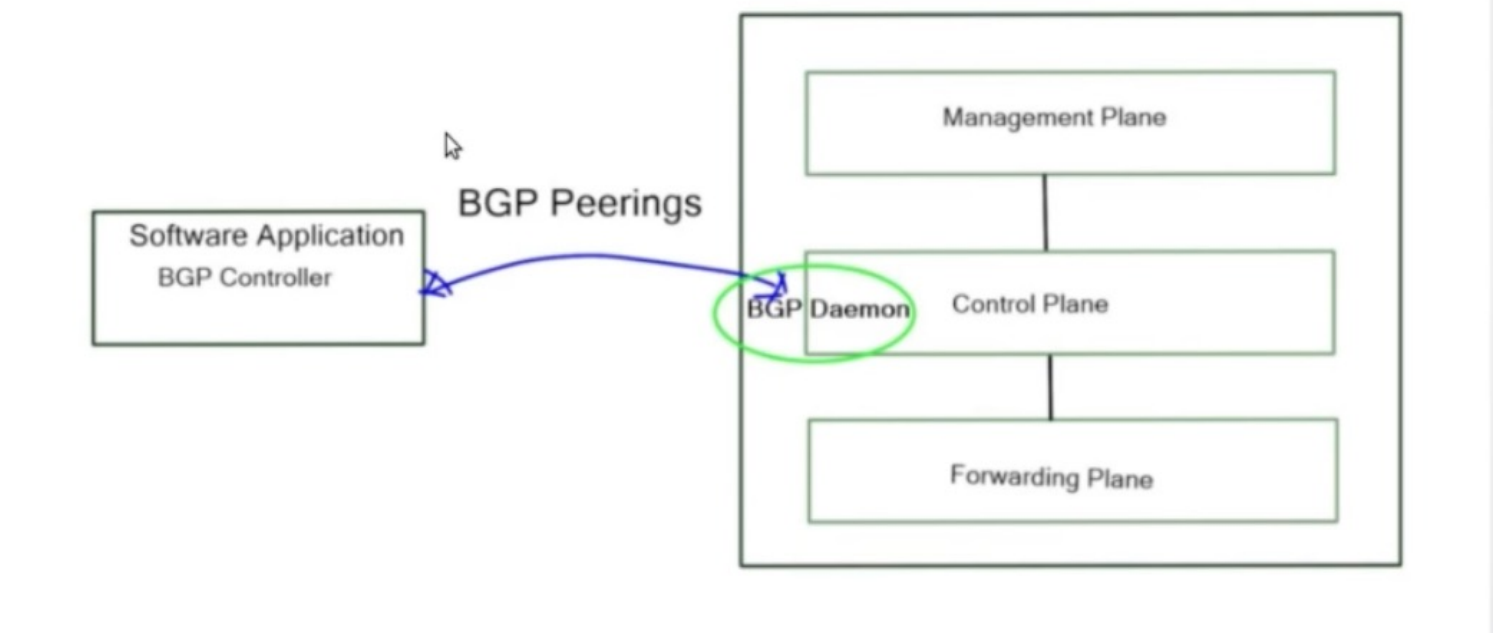
```
curl --user "admin":"admin" -H "Accept: application/json" -H "Content-Type: application/json" -X DELETE
http://localhost:8181/restconf/config/network-topology:network-
topology/topology/ovsdb:1/node/ovsdb:HOST2
```

After removed, check the status again.


SDN & BGP

- SDN's promises include **centralized network programming capabilities** which allow modern networks to forward, filter and/or classify flows based on encoded policies.
- Border Gateway Protocol (BGP) as the transfer protocol between the SDN controller and forwarding devices

BGP-Based SDN



Why BGP?

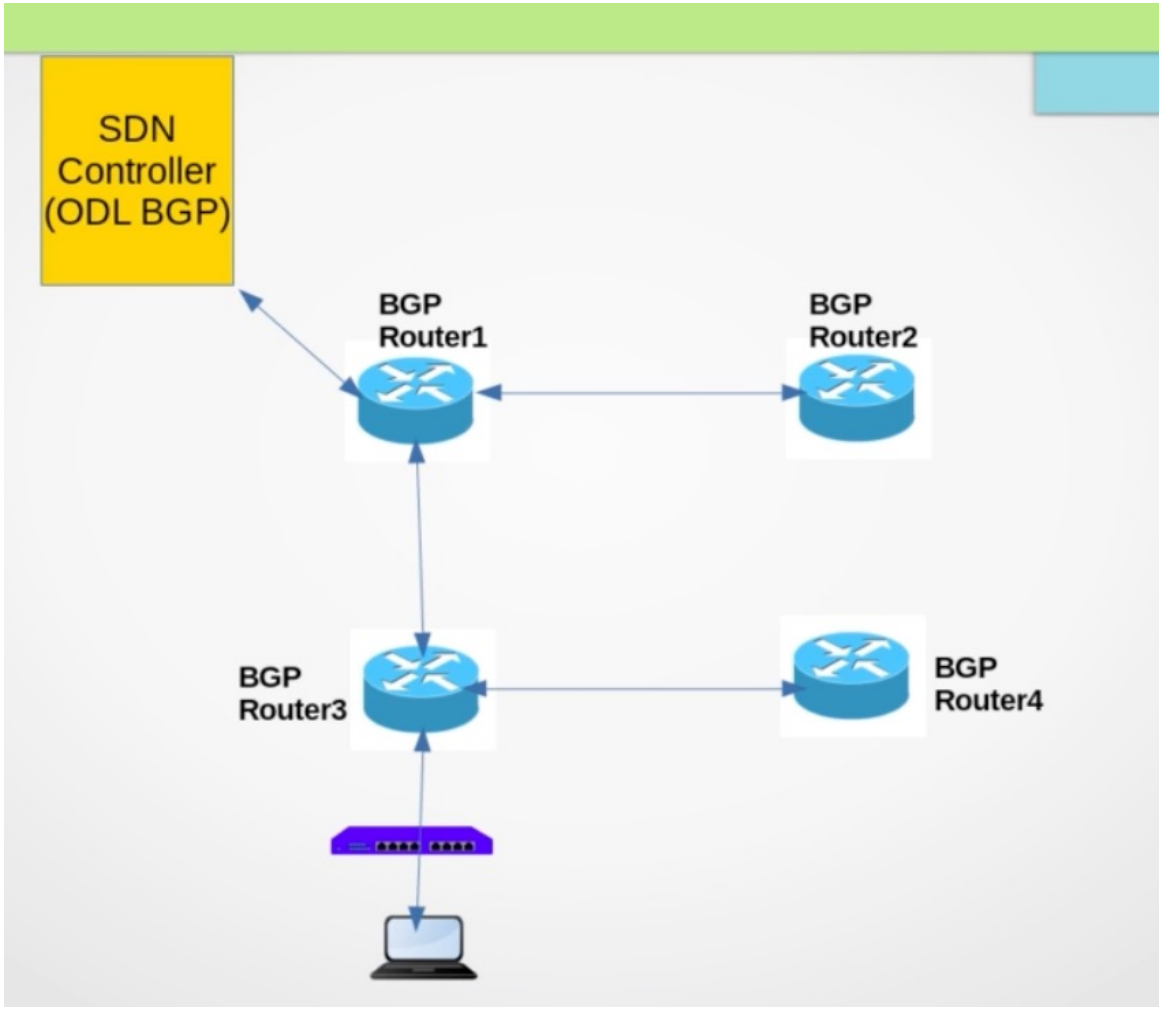
- The Border Gateway Protocol (BGP) is an inter-Autonomous System (AS) routing protocol. The primary role of the BGP is an exchange of routes among other BGP systems.
- BGP has so many extensions.
- can carry various types of routing information - L3VPN, L2VPN, IP multicast, linkstate(LS), flowspec etc.

Why BGP?

- BGP FlowSpec distributes forwarding entries, such as ACL and PBR to the TCAM of devices.
- L3VPN and EVPN offer the mechanism to integrate with legacy networks and service insertion.
- BGP-LS extracts IGP network topology information and passes it to the SDN controller via BGP updates..

SDN and BGP Usecases

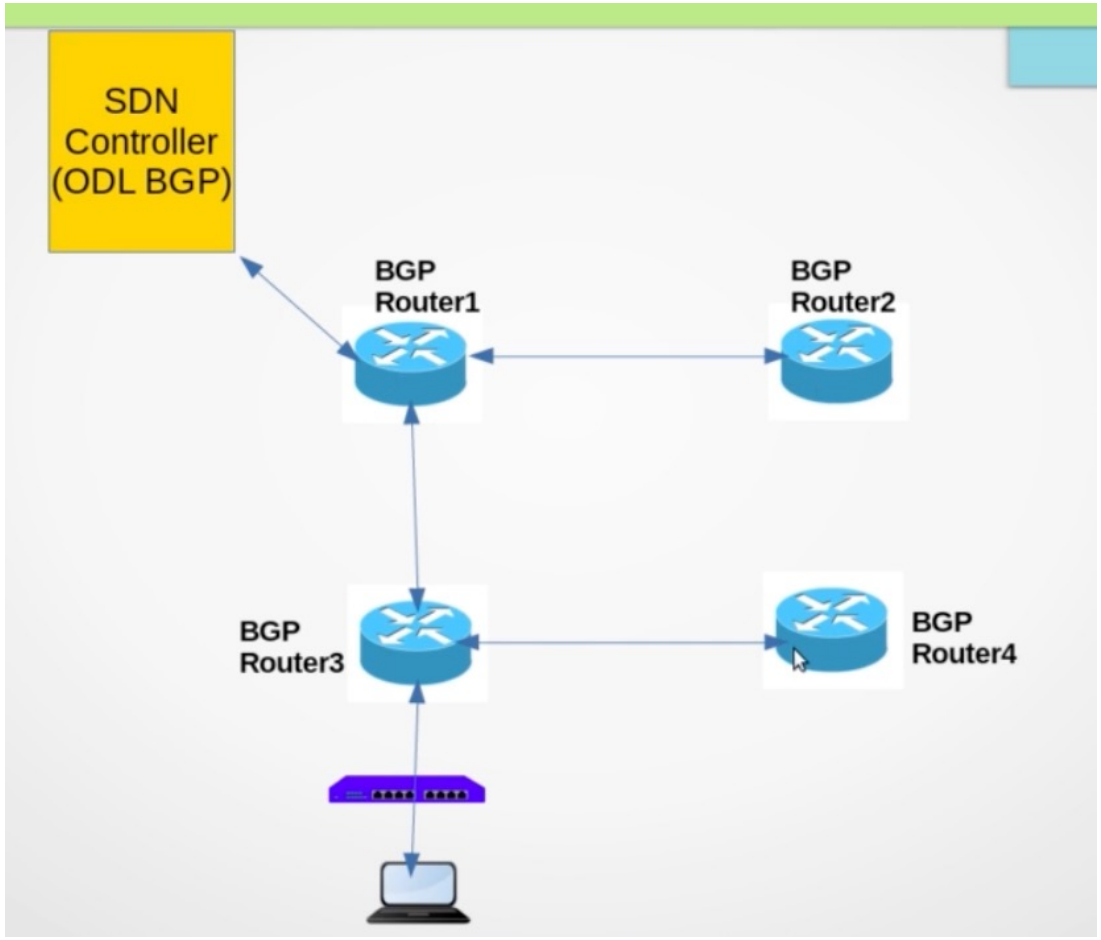
- SDN router - Turns switch into an Internet router
- Virtual Route Reflector - High-performance server-based BGP Route Reflector
- SDX - A Software Defined Internet Exchange controller
- Large-Scale Data Centers - BGP Data Center Routing, MPLS/SR in DCs, DC interconnection
- DDoS mitigation - Traffic Filtering distribution with BGP


SDN
Controller
(ODL BGP)
BGP
Router1
BGP
Router2
BGP
Router3
BGP
Router4

SDN & BGP Test Environment

Requirements

- Virtualized Test Environment like mininet.
- Devices - BGP Routers, switches, hosts
- Topology Builder (similar to mininet)
  - Links, ip address,
  - Multiple Routers, etc


SDN
Controller
(ODL BGP)
BGP
Router1
BGP
Router2
BGP
Router3
BGP
Router4

ContainerNet

- Containernet is a fork of the **Mininet**
- Allows to use **Docker** containers as hosts in network topologies
- Used in many NFV Usecases

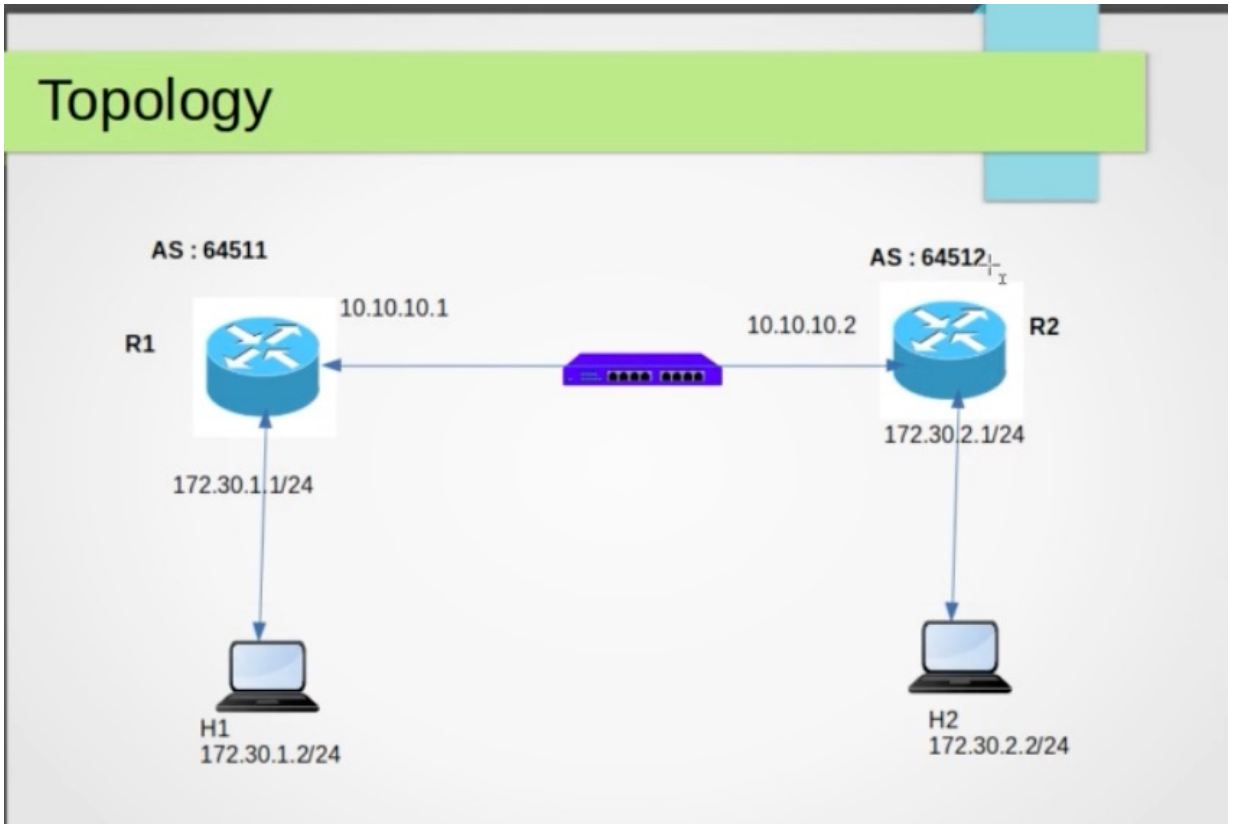
BIRD - BGP Router

- BIRD is an Internet Routing Daemon/Software , runs in UNIX
- Supports BGP, OSFP, RIP,etc
- We have built the docker with BIRD routing daemon (knet/urouter)

TestBed Installation

- System
  - Ubuntu 18.04 Virtual Machine
  - 2 Core / 8GB RAM
- ContainerNet Installation
- Quick Verification

Exercise I


Topology
AS : 64511
AS : 64512
10.10.10.1
10.10.10.2
R1
R2
172.30.1.1/24
172.30.2.1/24
H1
172.30.1.2/24
H2
172.30.2.2/24

ODL BGP

1. BGP Testbed(containernet) Installation

Containernet installation

Requires Ubuntu 18.04 LTS and Python3.

```
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
cd ..
```

Quick verification

```
sudo mn --controller=remote,ip=127.0.0.1 --topo=single,2
```

you should see the "containernet>" prompt

To see the docker stuff

```
sudo docker images
sudo docker ps -a
```

2. exercise1 - Traditional EBGP

How to run

i) Run the topology

```
sudo python3 topo1.py
```

ii) check the bgp protocol status of Router R1 and R2 in mininet shell

```
r1 birdc show protocols
r1 birdc show route
r1 ip route
```

R2

```
r2 birdc show protocols
r2 birdc show route
r2 ip route
```

BGP Peers are established between R1 and R2. Routes are synchronized.

Note: if you know docker commands, you can login/execute commands router docker .

Important Docker commands

```
sudo docker ps -a
sudo docker exec -it mn.r1 sh
```

birdc important commands

```
birdc show status
birdc show route
birdc show protocols
birdc show protocols all
```

Logs can be checked in

```
tailf /var/log/bird.log
```

iii) verify the ping traffic from h1 to h2

```
h1 ping h2
```

try traceroute also

```
h1 traceroute h2
```

3. OpendayLight & BGP Plugin Installation

- Java Installation
- ODL Installation
- Install the OpenFlow Plugin
  - *feature:install odl-bgpcep-bgp (ติดตั้งเพิ่มเติม)*

4. REST API Verifications

Create a BGP Instace

```
curl --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X POST
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/ -d @bgp_router.xml
```

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X GET
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/  | xmllint --format -
```

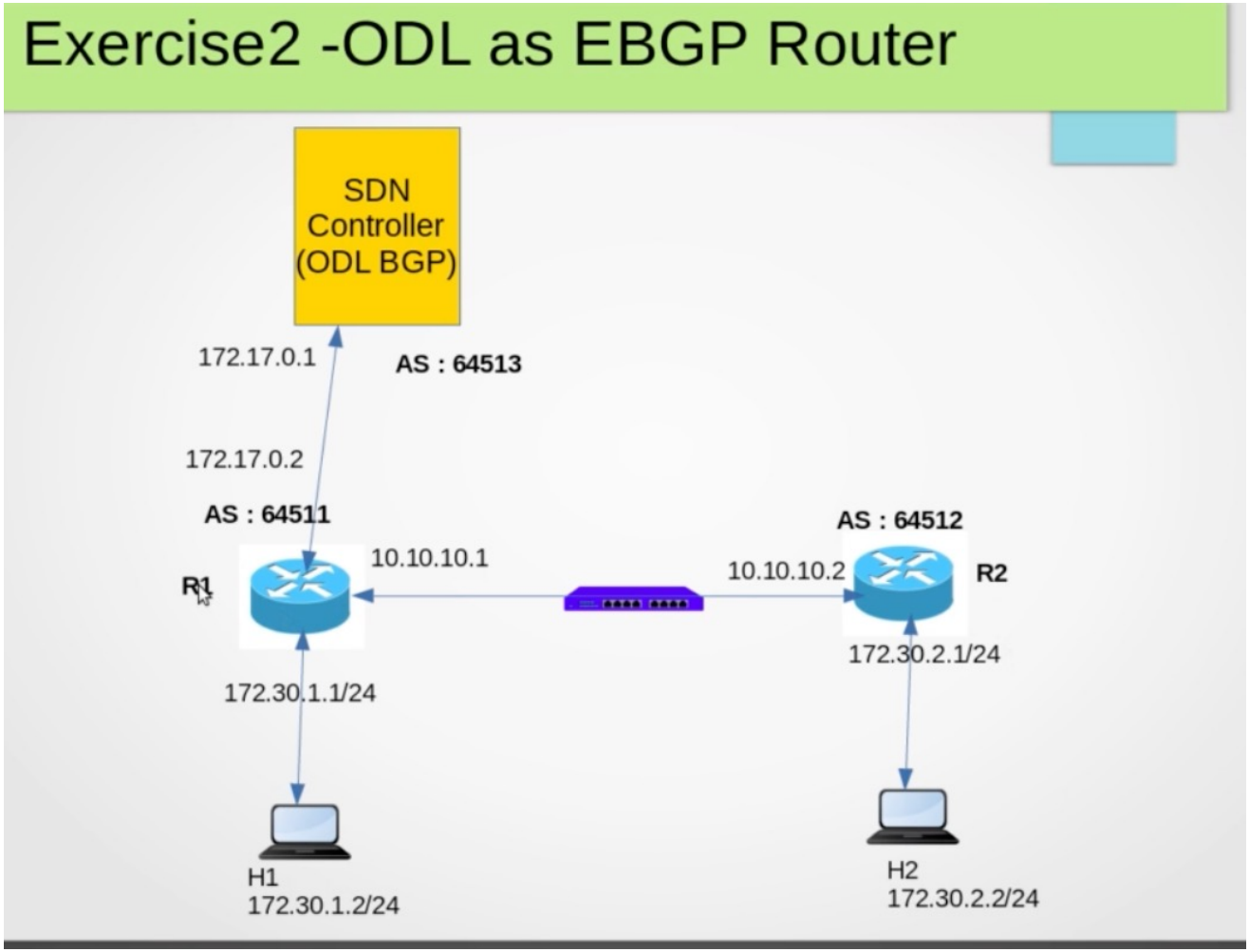
Verify the BGP Instace

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X  GET
http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/ | xmllint --format -
```

Delete the BGP Instance

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X
DELETE  http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/ -d @bgp_router.xml
```

Exercise II : ODL as EBGP Router



5. Exercise2 - ODL as EBGP router

i) **Run the topology**

```
sudo python3 topo1.py
```

ii) **check the bgp protocol status on Router R1 and R2**

R1

```
containernet>r1 birdc show protocols
containernet>r1 ip route
```

R2

```
containernet>r2 birdc show protocols
containernet>r2 ip route
```

BGP Peers are established between R1 and R2. Routes are synchronized.

iii) **Create the ODL BGP Instance**

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X POST
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/ -d @bgp_router.xml
```

iv) **verification**

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X  GET
http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/ | xmllint --format -
```

v) **create bgp neighbor**

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X  POST
http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/protocol/openconfig-policy-types:BGP/bgp-odl-
router/bgp/neighbors/ -d @bgp_neighbor.xml
```

vi) **verify bgp neighbor/route details**

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X  GET
http://localhost:8181/restconf/operational/bgp-rib:bgp-rib/rib/bgp-odl-router/ | xmllint --format -
```

vii) **verify the bgp ports and neighbor establishment**

```
containernet>r1 birdc show protocols
containernet>r1 ip route
```

```
$ netstat -a | grep bgp
tcp        0      0 172.17.0.1:50876        172.17.0.2:bgp          ESTABLISHED
```

vii) **Destroy the test**

Delete the BGP instance in ODL

```
curl -v --user "admin":"admin" -H "Accept: application/xml" -H "Content-Type: application/xml" -X
DELETE  http://localhost:8181/restconf/config/openconfig-network-instance:network-instances/network-
instance/global-bgp/openconfig-network-instance:protocols/ -d @bgp_router.xml
```

exit from containernet.