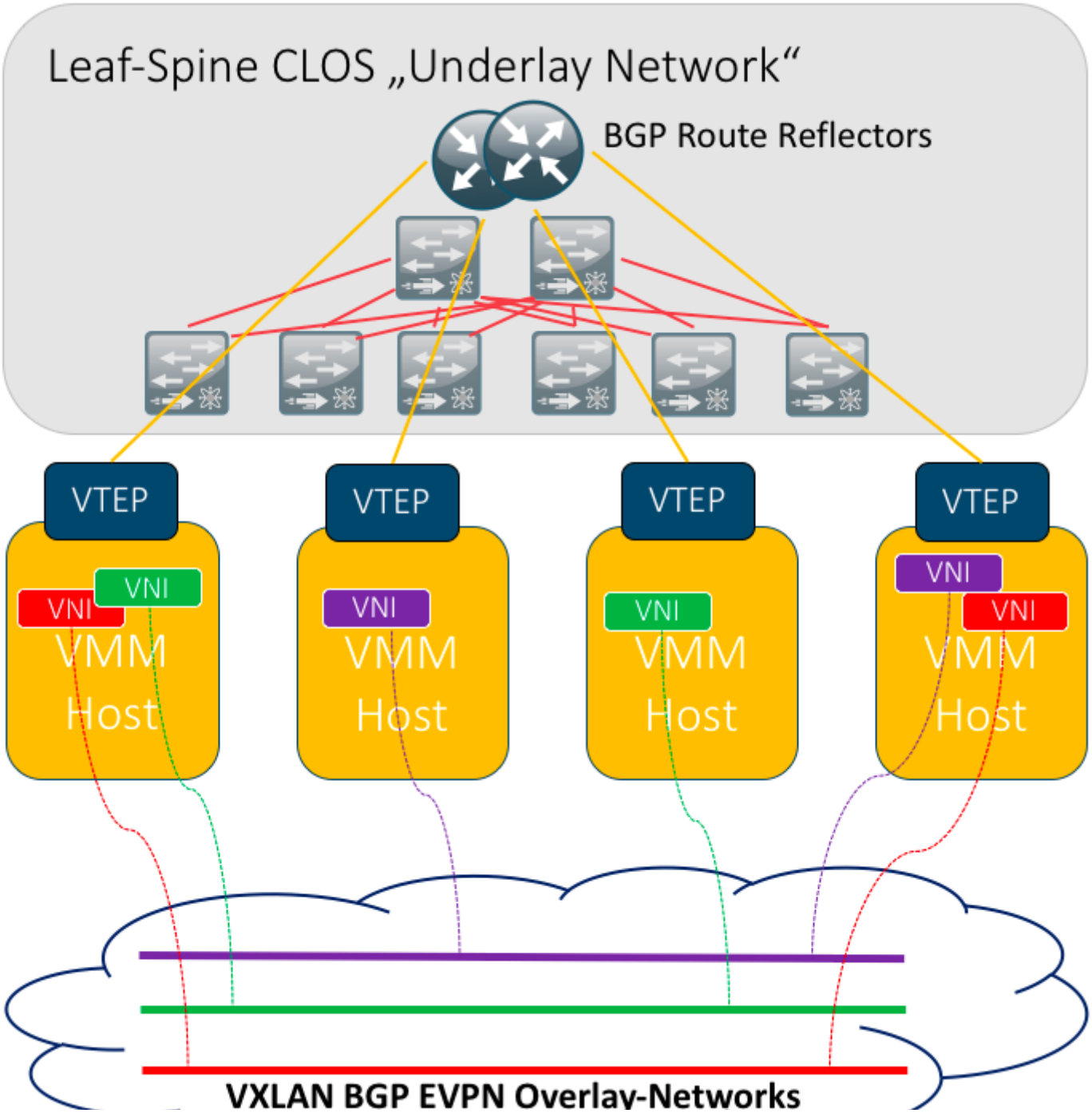
CLOUD BLOG – OPENSOURCE POWER @ENTERPRISE & DATACENTER

Infrastructure-as-a-Service / Platform-as-a-Service / Datacenter 2.0


Leaf-Spine CLOS „Underlay Network“
BGP Route Reflectors
VTEP
VTEP
VTEP
VTEP
VNI
VNI
VNI
VNI
VNI
VNI
VMM Host
VMM Host
VMM Host
VMM Host
VXLAN BGP EVPN Overlay-Networks

13. JULI 2017 VON SEBASTIAN

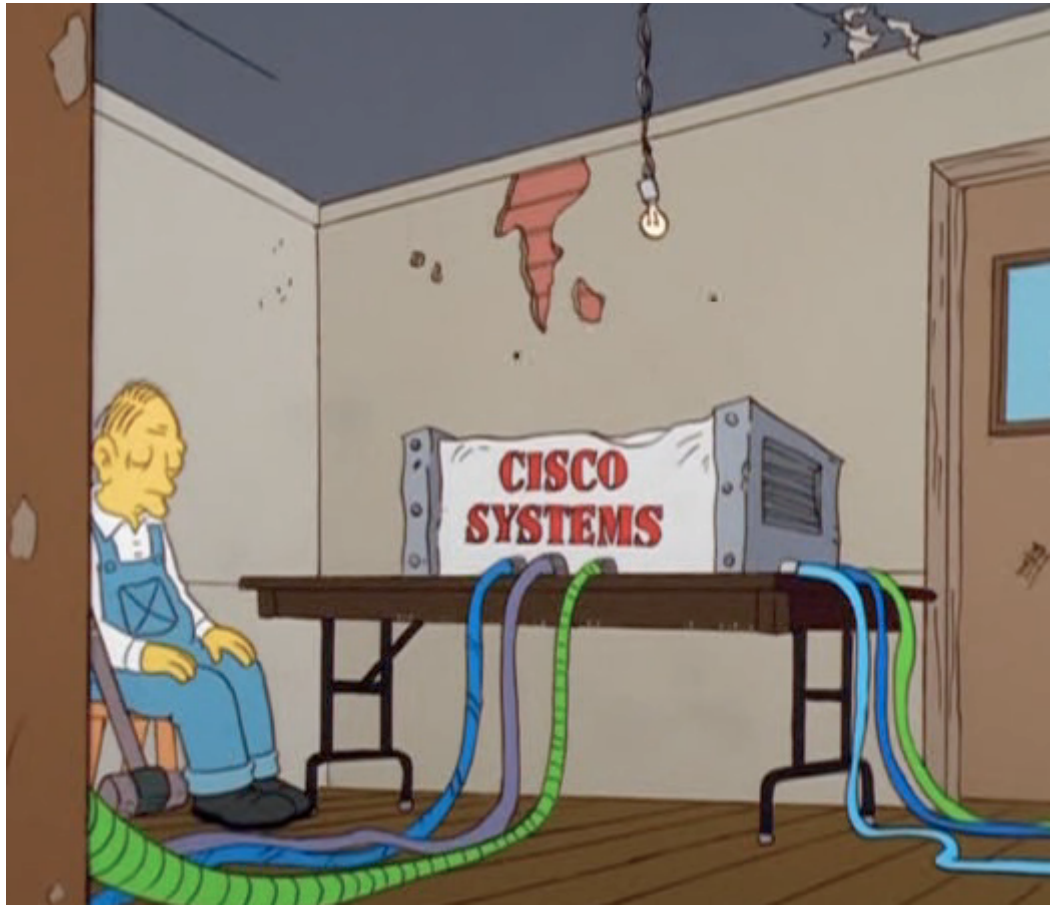
BGP EVPN (MP-BGP + VXLAN) with OpenNebula

From my point of view a well designed datacenter fabric based on VXLAN with BGP EVPN as a kind of control plane, based on a Leaf-Spine (CLOS) topology is one of the secrets to gain highly scalable, resilent and performant multi-tenancy for datacenter and cloud networks.

Thanks to Mario's idea i started playing around with this fancy technology. And of course, my employer NTS is a networking company – and last but not least: it's SDN and that's cool 🙂

In this blog post we try to build a lab setup with BGP-EVPN and OpenNebula. Perhaps i will go to develop an OpenNebula Add-On or VNM-Driver if it`s needed.

Because it's inside our Lab i'm going to implement the BGP part with Docker and „Cumulus Quagga", a software BGP implementation. In a real world scenario the Route Reflectors can be the Cisco ToR switches or any Leaf/Spine (Nexus) switch. Cause we use Standards, the implementation is open for any other vendor. But of course, there is only one vendor :



Why BGP EVPN / MP BGP with VXLAN?

With BGP EVPN / MP-BGP as control plane for VXLAN we will work with proven and well-kown standards – BGP empowers the whole internet. It`s capable to handle thousands and thousands of routes with ease.

The following RFCs and Drafts are taken into account for this solution:

- VXLAN (RFC7348)
- BGP4 (RFC4271)

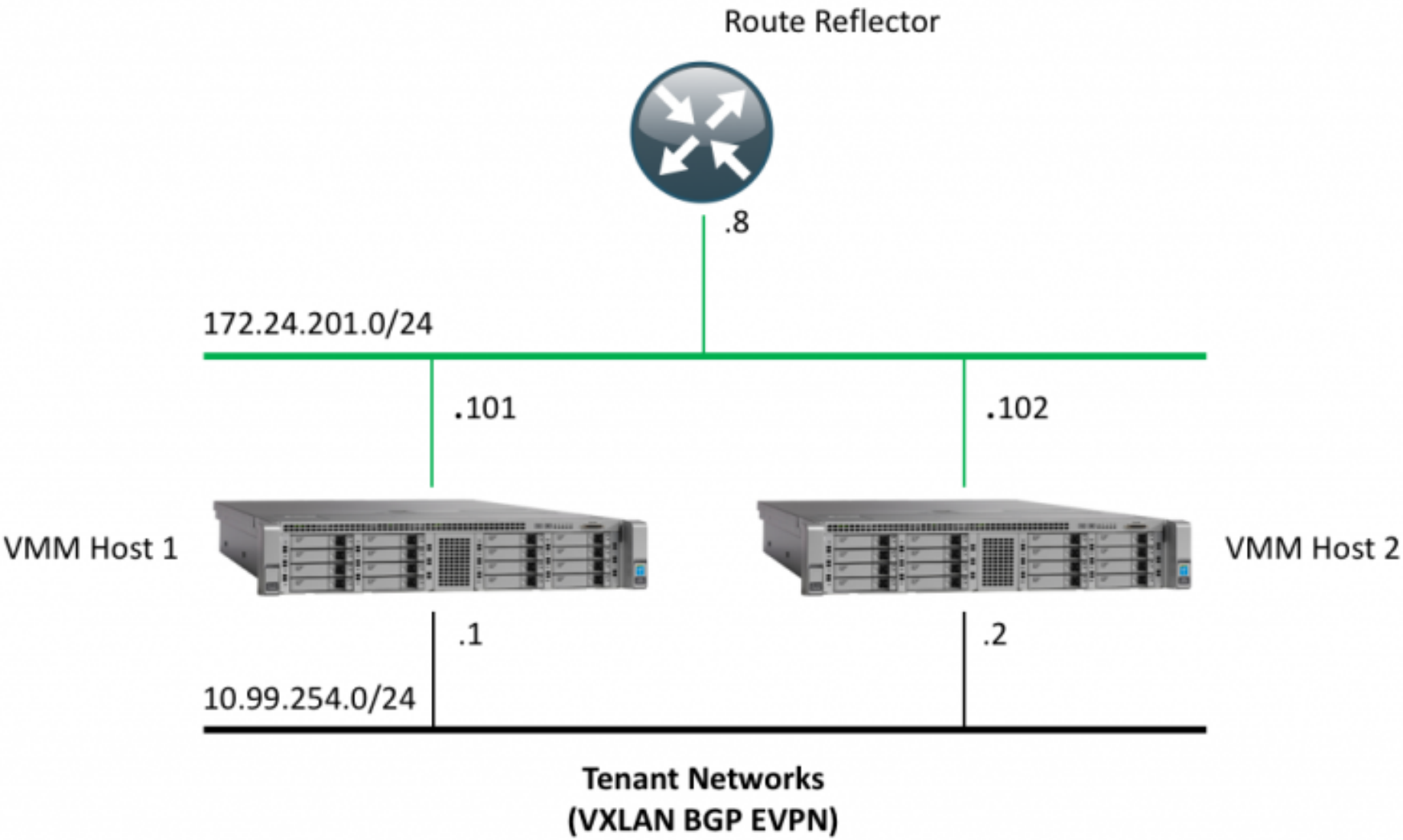
- BGP MPLS-Based Ethernet-VPN (RFC 7432)
- draft-ietf-bess-evpn-overlay-08
- MP-BGP (RFC4760)

VXLAN with BGP EVPN has several advantages over VXLAN with Multicast:

- Traffic engineering
- More scalability
- Better manageability
- Easier to handle than IP-Multicast in a large scale network

In our case BGP carries the reachability informations (f.e. the forwarding tables of our virtual switches/bridges) in Network Layer Reachability Information (NLRI) format for our EVPN. In particular it carries the MAC-Adresses and the information, where they are attached to (VTEP).

Our lab setup looks like this:

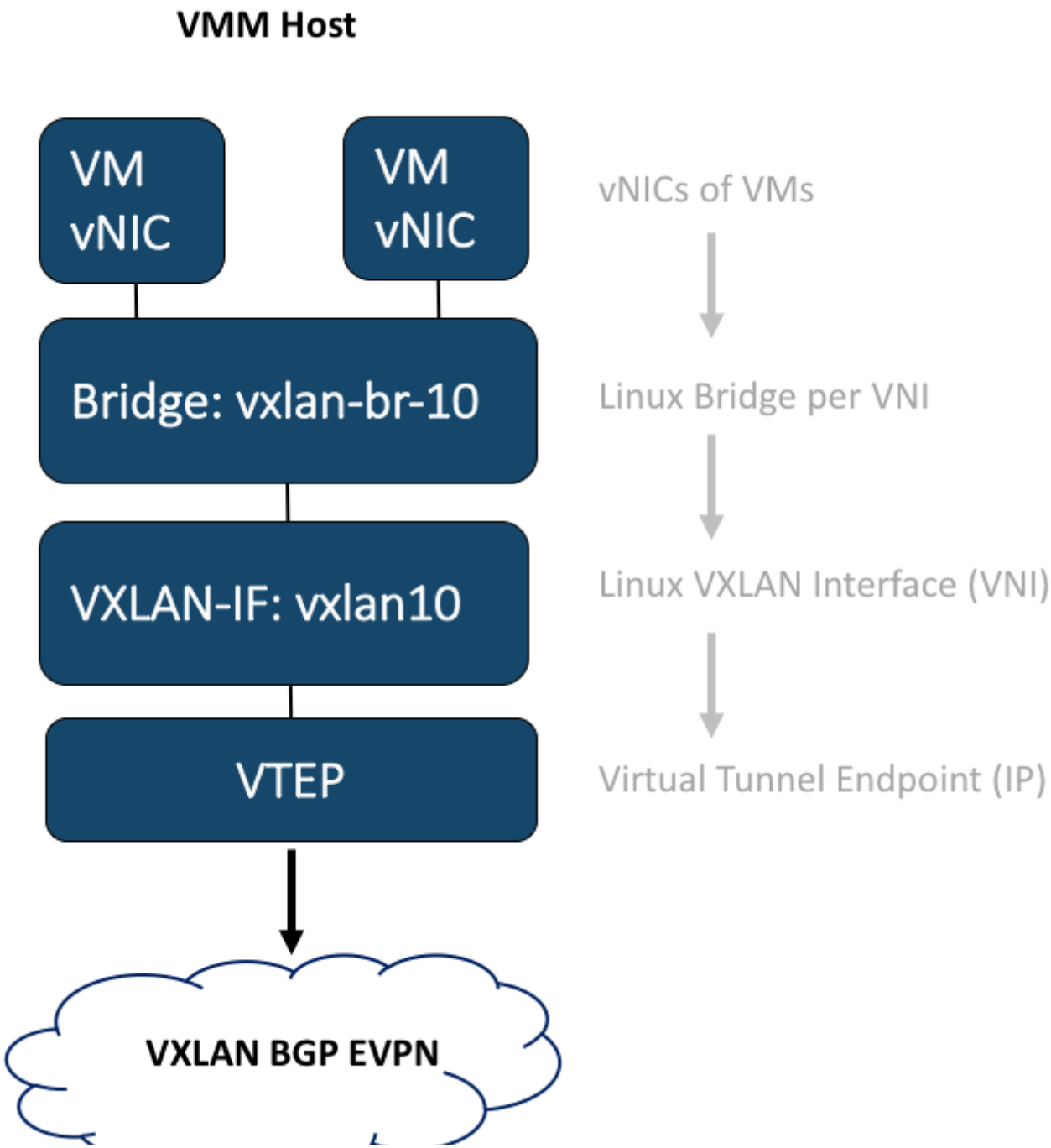


VXLAN BGP EVPN Lab network

Every VTEP (in our case the hypervisor host) exports two kinds of „routes“ via BGP:

- Type 2 Route: The local MAC adresses for the VNI
- Type 3 Route: The local VNIs with it`s IPv4-Endpoint – the VTEP

The networking stack inside the hypervisor hosts is based on proven and well known standards (vNICs of the VMs, attached to an Linux-Bridge and a VXLAN-Interface for each VNI enslaved to that bridge):


VMM Host
VM vNIC
VM vNIC
Bridge: vxlan-br-10
VXLAN-IF: vxlan10
VTEP
VXLAN BGP EVPN
vNICs of VMs
Linux Bridge per VNI
Linux VXLAN Interface (VNI)
Virtual Tunnel Endpoint (IP)

Networking Stack inside a VMM Host

In our case we operate one Route-Reflector, also Docker based, on a VM in our lab. In a real world example we`ll run at least two Route-Reflectors. The configuration of our Route-Reflector is quite simple:

- Create a peer-group called „vxlan“
- Create a dynamic membership for the VTEP network (172.24.201.0/24)
- Reflect all EVPN to all clients

First we had to pull the current Docker-Image for the Quagga implementation and start it:

```
1.  docker pull cumulusnetworks/quagga
2.
3.  # Quagga has to run in "privileged" mode to get the attached VNIs of the VMM hosts and to inject the FDBs
4.  docker run -t -d --net=host --privileged --name BGP-EVPN-RR1 cumulusnetworks/quagga:latest
```

Now we are able to configure our RR:

```
1.  docker exec -i -t BGP-EVPN-RR1 /usr/bin/vtysh
```

The CLI is like our well known Cisco CLI 🙂

```
1.  router bgp 64512
2.    bgp router-id 172.24.201.8
3.    bgp cluster-id 172.24.201.8
4.    bgp log-neighbor-changes
5.    no bgp default ipv4-unicast
6.    neighbor vxlan peer-group
7.    neighbor vxlan remote-as 64512
8.    neighbor vxlan capability extended-nexthop
```

```
  neighbor vxlan update-source 172.24.201.8
  bgp listen range 172.24.201.0/24 peer-group vxlan
  !
  address-family evpn
   neighbor vxlan activate
   neighbor vxlan route-reflector-client
  exit-address-family
  !
```

At the VTEP side we also had to run a BGP implementation, wich advertises all connected VNIs to the Route-Reflector:

```
router bgp 64512
 bgp router-id 172.24.201.102
 no bgp default ipv4-unicast
 neighbor vxlan peer-group
 neighbor vxlan remote-as 64512
 neighbor vxlan update-source vmnet0
 neighbor vxlan capability extended-nexthop
 neighbor 172.24.201.8 peer-group vxlan
 !
 address-family evpn
  neighbor vxlan activate
  advertise-all-vni
 exit-address-family
!
```

From now, all VNIs will be advertised from the VTEP to the RR. The RR will reflect all EVPN to all connected neighbours. Of course there are much more features (policys, communities, …) – i cannot cover them, because i`m not the „networking guy“ – that’s stuff for my brilliant networking colleagues like Mario.

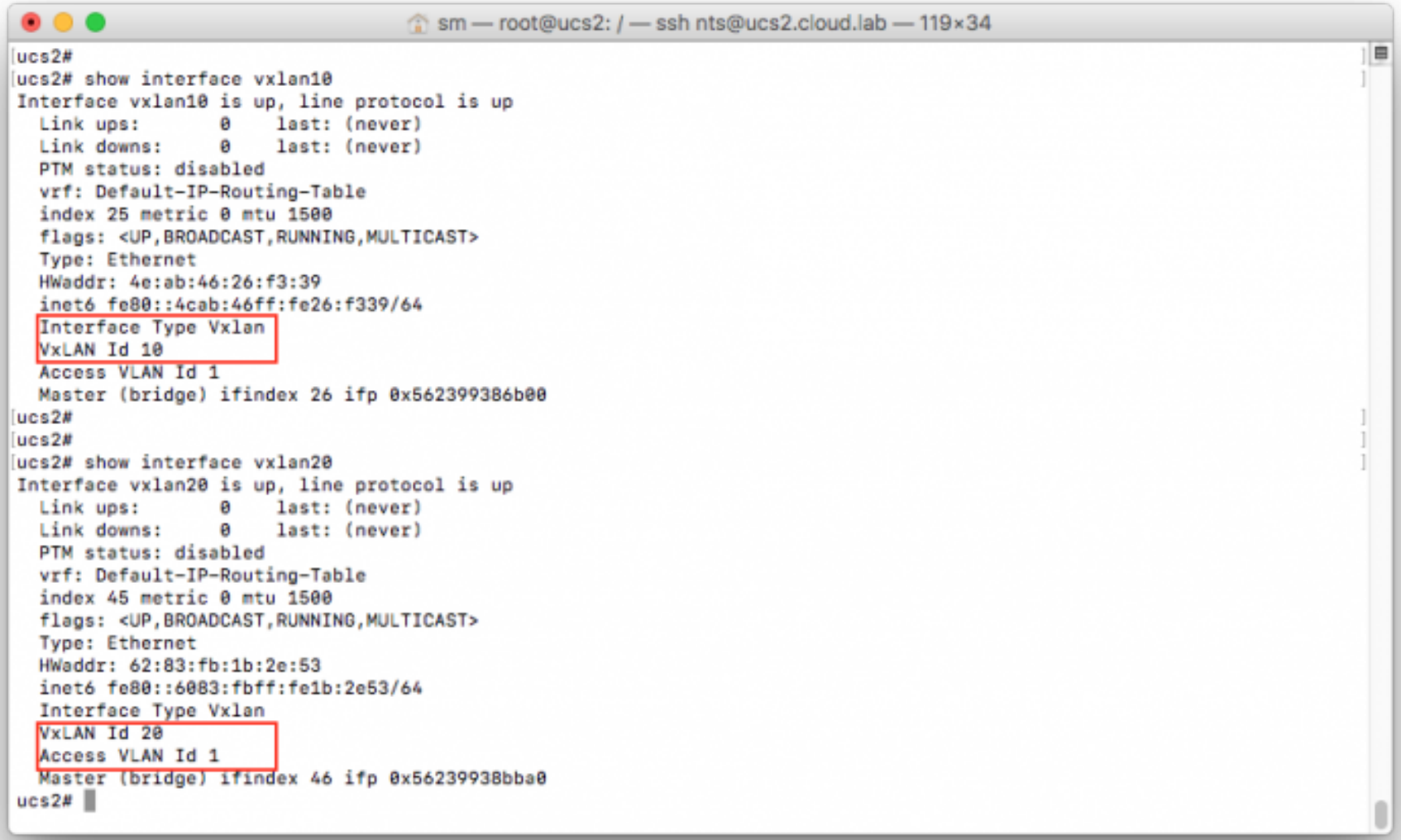
To create some VNIs at our VMM hosts i crafted a litte script to create the VNI, VTEP and a bridge to attach the VNICs of the VMs on it:

```
#!/bin/bash

# 2017 NTS / S.Mangelkrmaer
# BGP EVPN VXLAN Lab

for ((vni = 10 ; vni <= 20 ; vni += 1)); do

  echo "Creating VXLAN Network with VNI-ID: $vni"

  # VTEP IP of physical iface eth3
  vtep=`ip -4 addr show eth3 | grep -Po 'inet \K[\d.]+'`
  ip link add vxlan${vni} type vxlan id ${vni} dstport 4789 local $vtep nolearning

  # Create Bridge
  brctl addbr vxlan-br-${vni}

  # Add VTEP to Bridge
  brctl addif vxlan-br-${vni} vxlan${vni}

  # Disable STP
  brctl stp vxlan-br-${vni} off

  # Set links up
  ip link set up dev vxlan-br-${vni}
  ip link set up dev vxlan${vni}
done
```

Within our OpenNebula we simply create new „Bridged Networks“.

In this example we will use two VXLAN networks „vxlan10“ and „vxlan20“, create 4 virtual machines on two different VMM hosts and start our testing.

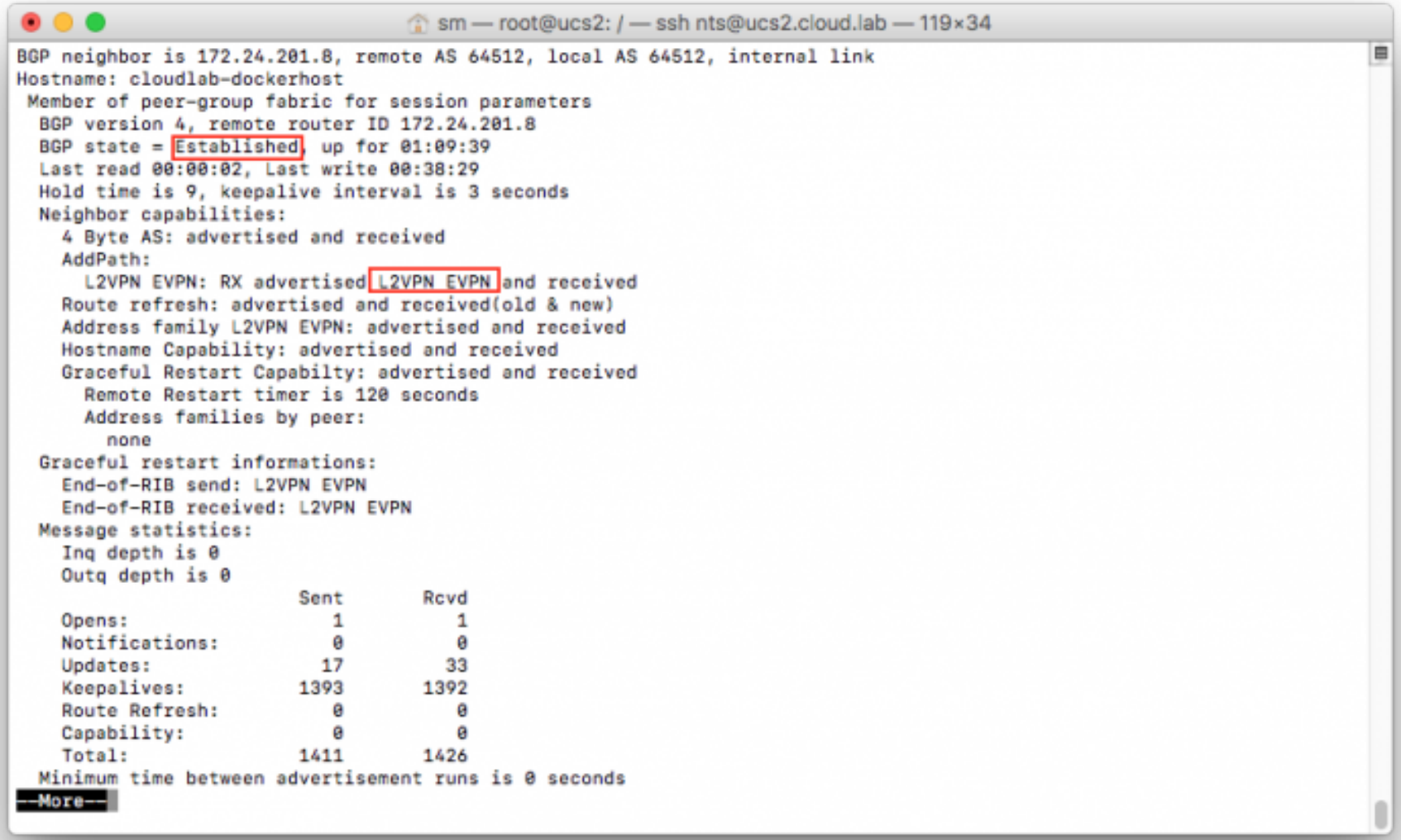
After instantiating the VMs we will check if we can see the VXLAN networks at the VTEP – this can be done inside our Quagga-Docker-Container with the `show interface $VNI` command:



Show VXLAN interface at VTEP

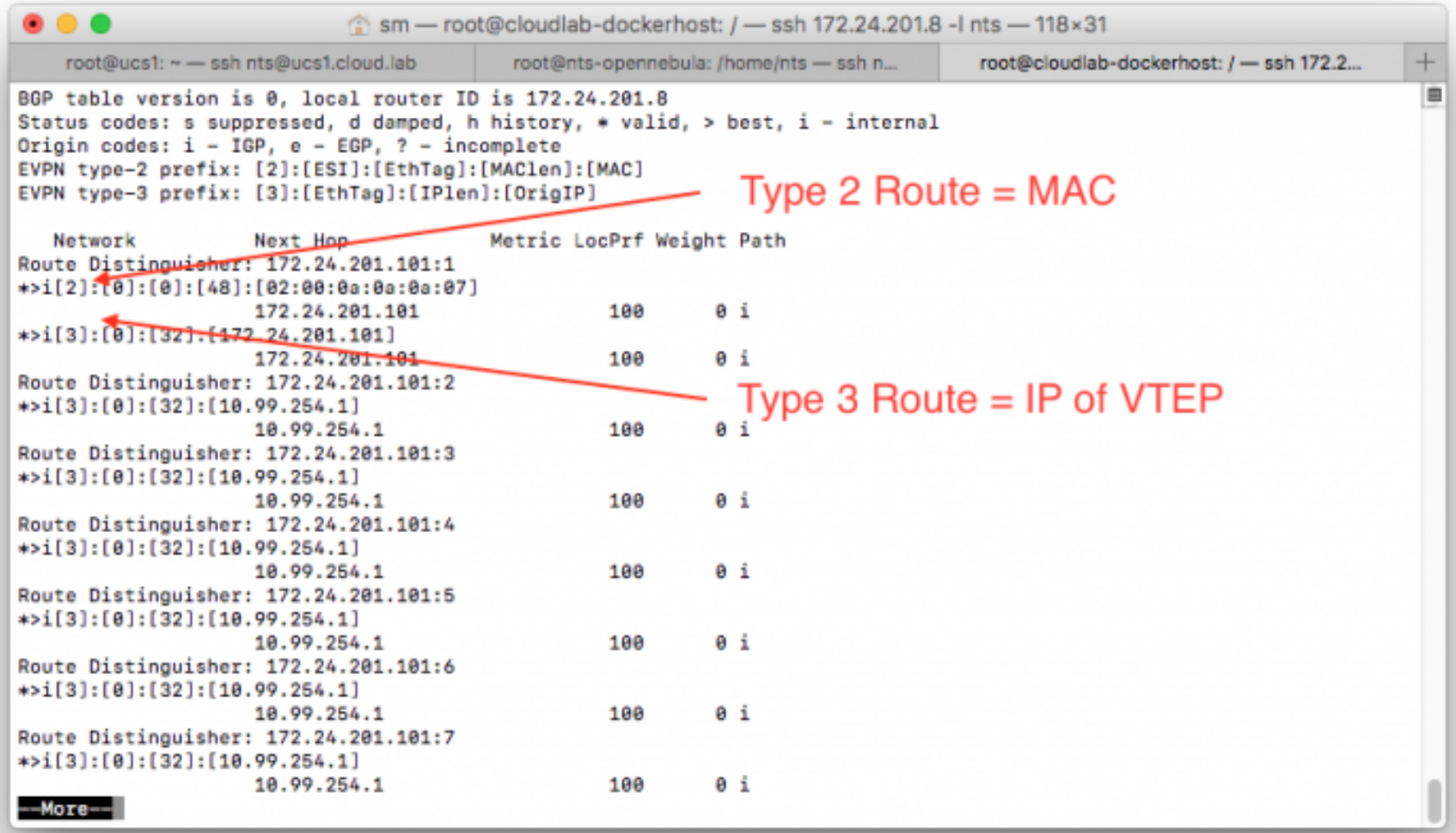
We see the VNI and the correct detected linux bridge (where the vNICs are attached to). This looks good.

Next we check if the VTEPs correctly has established their BGP session to the RR with the `show bgp neighbours` command:



Show BGP neighbours

At our RR there has to be a Type 3 route for every VNI and a Type 2 route for every MAC inside the VNI of a VMM host:



Show BGP EVPN routes at RR

The syntax of the Type 2 and 3 route is like this:

- **Type 2 Route :** `[2]:[ESI]:[EthTag]:[MAClen]:[MAC]`
- **Type 3 Route:** `[3]:[EthTag]:[IPlen]:[OrigIP]`

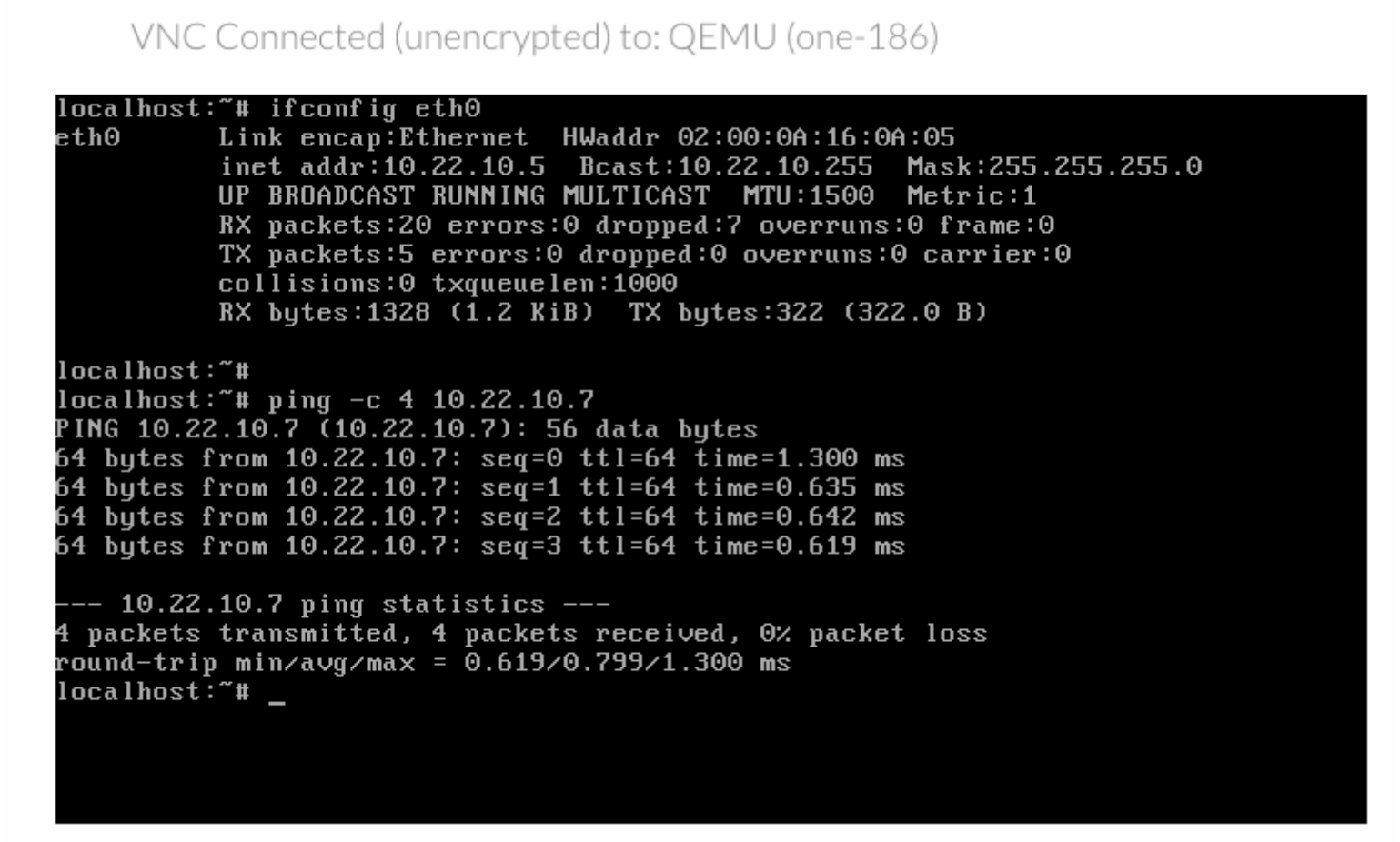
Now we're able to run some checks. First we start a „PING from the first VM (running on the first VMM Host) to the second VM (running on the secondary VMM Host). Then we will check the Forwarding Tables of our „virtual switches" – the Linux Bridged with their VXLAN interfaces attached to.

We will ping from SRC_IP: 10.22.10.5 to DST_IP: 10.22.10.7 – see the details of the VMs below:

ID	Owner	Group	Name	Status	Host	IPs
187	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-10-3	RUNNING	ucs1.cloud.lab	10.22.10.7
186	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-10-1	RUNNING	ucs2.cloud.lab	10.22.10.5
185	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-10-2	RUNNING	ucs1.cloud.lab	10.22.10.6
184	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-10-0	RUNNING	ucs2.cloud.lab	10.22.10.4
183	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-20-1	RUNNING	ucs1.cloud.lab	10.22.20.4
182	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-20-2	RUNNING	ucs2.cloud.lab	10.22.20.5
181	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-20-0	RUNNING	ucs1.cloud.lab	10.22.20.6
180	oneadmin	oneadmin	vxlan-bgp-evpn-test-vni-20-3	RUNNING	ucs2.cloud.lab	10.22.20.7

VM Details

The result looks good – as expected:



ICMP PING

Let's dig a bit deeper down the layers. First let's have a look to the FDBs of the VMM hosts:

FDB VNI 10 @VMM-HOST "UCS1"

```
1.  # bridge fdb show dev vxlan10
2.
3.  12:37:89:6b:a3:49 master vxlan-br-10 permanent
4.  02:00:0a:16:0a:05 master vxlan-br-10
5.  02:00:0a:16:0a:05 vlan 1 master vxlan-br-10
6.  12:37:89:6b:a3:49 vlan 1 master vxlan-br-10 permanent
```

```
00:00:00:00:00:00 dst 10.99.254.2 self permanent
02:00:0a:16:0a:05 dst 10.99.254.2 self offload
```

FDB VNI 10 @VMM-HOST "UCS2"

```
# bridge fdb show dev vxlan10

02:00:0a:16:0a:07 master vxlan-br-10
4e:ab:46:26:f3:39 vlan 1 master vxlan-br-10 permanent
4e:ab:46:26:f3:39 master vxlan-br-10 permanent
02:00:0a:16:0a:07 vlan 1 master vxlan-br-10
00:00:00:00:00:00 dst 10.99.254.1 self permanent
02:00:0a:16:0a:07 dst 10.99.254.1 self offload
```

Our SRC-VM has the MAC „02:00:0a:16:0a:05“ and runs on VMM host „UCS2“, which has the VTEP-IP „10.99.254.2“. The Forwaring Database of our VXLAN bridge at the VMM host „UCS1“ shows a entry for this MAC address pointing to the VTEP IP of the VMM host „UCS2“:

```
02:00:0a:16:0a:05 dst 10.99.254.2 self offload
```

The DST-VM ha the MAC „02:00:0a:16:0a:07“ and runs on VMM host „UCS1“, which has the VTEP-IP „10.99.254.1“. The entry in the FDB of VMM host „UCS2“ looks like this:

```
02:00:0a:16:0a:07 dst 10.99.254.1 self offload
```

The „00:00:00:00:00:00 “ MAC is the „Default-Route“ for any BUM-Traffic.

Looks good!

This information (Type 2 and Type 3 Routes) are distributed by our BGP EVPN. Let's check this layer.

First on our VMM host „UCS1".

Display some information about our VNI 10:

```
ucs1# sh evpn vni 10

VNI: 10
 VxLAN interface: vxlan10 ifIndex: 20 VTEP IP: 10.99.254.1
 Remote VTEPs for this VNI:
  10.99.254.2
 Number of MACs (local and remote) known for this VNI: 2
 Number of ARPs (IPv4 and IPv6, local and remote) known for this VNI: 0
```

The interesting information is our local VTEP IP and a list of the remote VTEPs for this VNI. Also there is some information about the MAC count inside the VNI.

To display more details we use the command „sh evpn mac vni 10":

```
ucs1# sh evpn mac vni 10

Number of MACs (local and remote) known for this VNI: 2
MAC               Type   Intf/Remote VTEP      VLAN
02:00:0a:16:0a:05 remote 10.99.254.2
02:00:0a:16:0a:07 local  one-187-0
```

Now we can close the circle to our FDBs 🙂

And – of course – it's BGP we also took a look to the routing information:

```
ucs1# sh bgp evpn route  vni 10
```

```
2.
3.  BGP table version is 0, local router ID is 172.24.201.101
4.  Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
5.  Origin codes: i - IGP, e - EGP, ? - incomplete
6.  EVPN type-2 prefix: [2]:[ESI]:[EthTag]:[MAClen]:[MAC]
7.  EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
8.
9.     Network          Next Hop            Metric LocPrf Weight Path
10. *>i[2]:[0]:[0]:[48]:[02:00:0a:16:0a:05]
11.                    10.99.254.2              0    100      0 i
12. *> [2]:[0]:[0]:[48]:[02:00:0a:16:0a:07]
13.                    10.99.254.1                       32768 i
14. *> [3]:[0]:[32]:[10.99.254.1]
15.                    10.99.254.1                       32768 i
16. *>i[3]:[0]:[32]:[10.99.254.2]
17.                    10.99.254.2              0    100      0 i
18.
19. Displayed 4 prefixes (4 paths)
```

This is only a short, quick and dirty blog post about this impressive technology. Perhaps there are some projects for implementations like this. But that's a part, i',m not allowed to write it down here.

ALLGEMEIN, OPENNEBULA