Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

1. Une trame Ethernet est créée par une machine dans un VLAN.
2. Cette trame est encapsulée dans un paquet **VXLAN**.
3. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
4. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
5. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

1. Le VTEP reçoit la **trame Ethernet**
2. Il ajoute un **header VXLAN**
3. Il encapsule le tout dans **UDP + IP**
4. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

1. Il retire l'en-tête **IP**
2. Il retire l'en-tête **UDP**
3. Il retire l'en-tête **VXLAN**
4. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

1. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
2. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
3. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	rôle
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

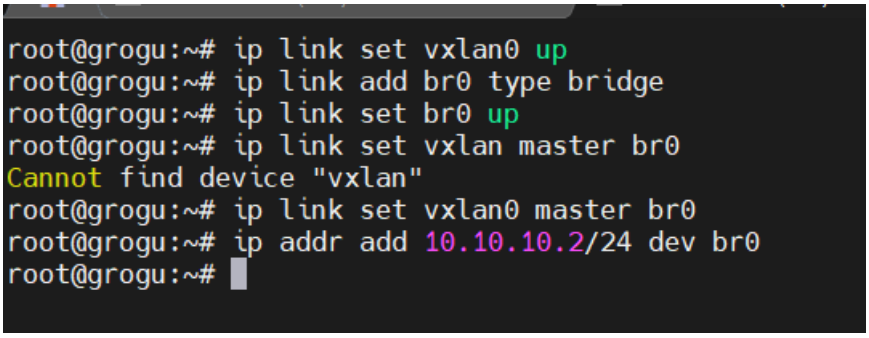
ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :



Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l'apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

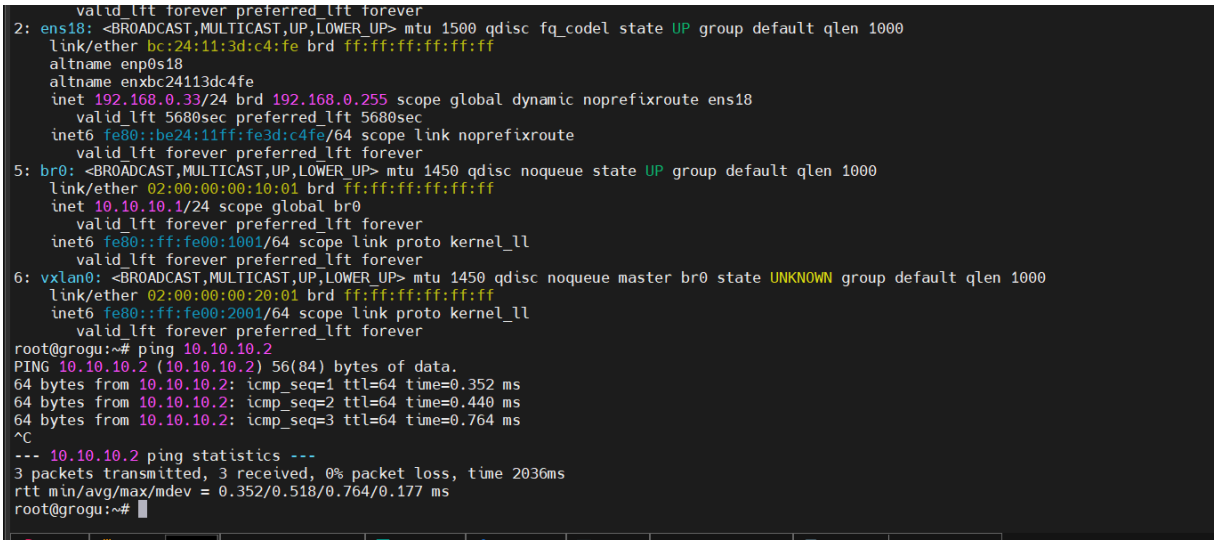
```
root@grogu:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.239 ms
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.239 ms (DUP!)
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.355 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=0.366 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=0.369 ms
^C
--- 10.10.10.2 ping statistics ---
4 packets transmitted, 4 received, +1 duplicates, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.239/0.313/0.369/0.061 ms
root@grogu:~#
```

Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

ip link set ens18 master br0

```
ip link set ens18 master br0
-bash: Sur : commande introuvable
root@grogu:~# ip link set ens18 master br0
root@grogu:~#
```

VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Mar  6 16:29:45 2026
root@VM-DOCKER:~# ip addr add 10.10.10.101 dev ens18
root@VM-DOCKER:~#
```

Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19

```
root@grogu:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:3d:c4:fe brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc24113dc4fe
    inet 192.168.0.33/24 brd 192.168.0.255 scope global dynamic noprefixroute ens18
       valid_lft 7007sec preferred_lft 7007sec
    inet 192.168.0.41/24 brd 192.168.0.255 scope global secondary dynamic ens18
       valid_lft 7193sec preferred_lft 7193sec
    inet6 fe80::be24:11ff:fe3d:c4fe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
5: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    link/ether 02:00:00:00:10:01 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.1/24 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::ff:fe00:1001/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br0 state UNKNOWN group default qlen 1000
    link/ether 02:00:00:00:20:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ff:fe00:2001/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:5e:f4:06 brd ff:ff:ff:ff:ff:ff
    altname enp0s19
    altname enxbc24115ef406
    inet 192.168.0.40/24 brd 192.168.0.255 scope global dynamic ens19
       valid_lft 7193sec preferred_lft 7193sec
root@grogu:~#
```

Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

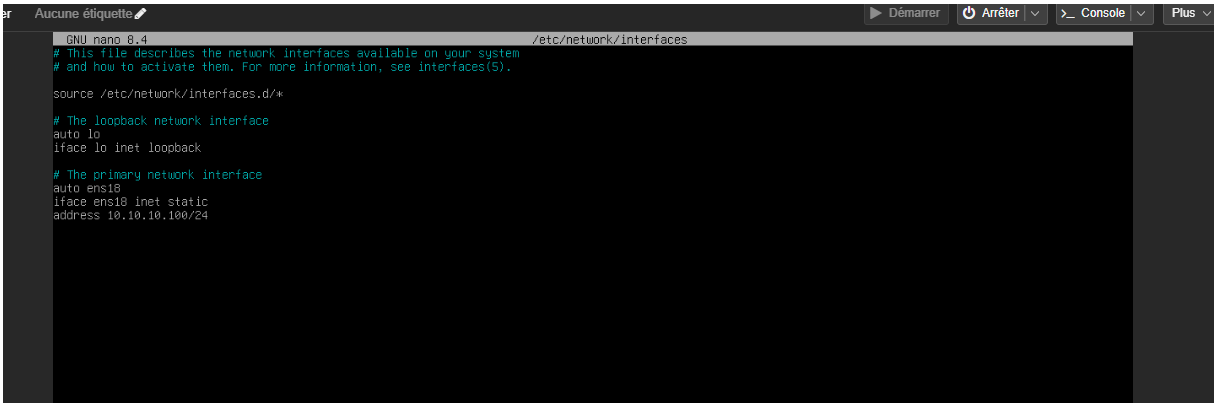
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

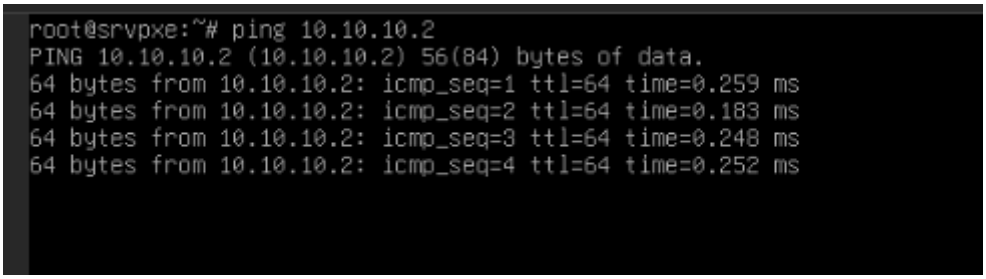
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

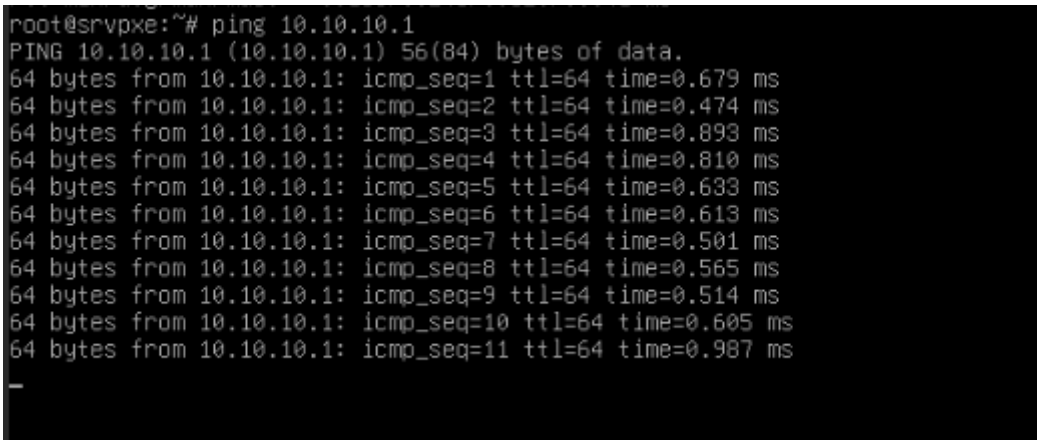
Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

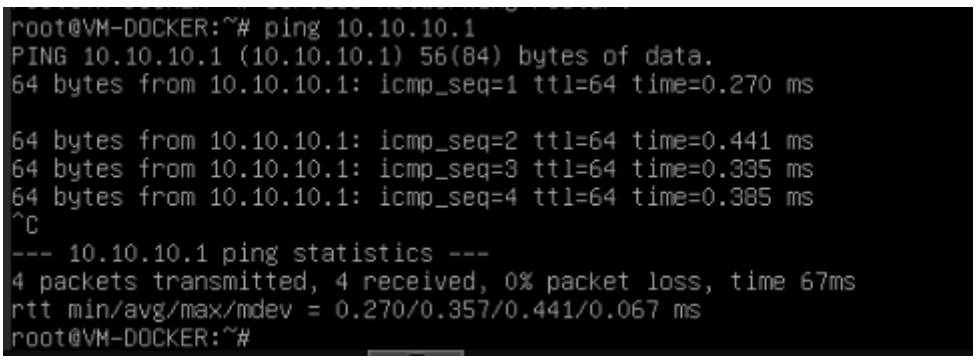
Il passe aussi avec succès



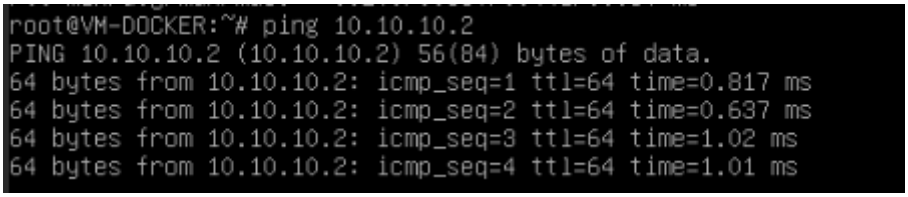
Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

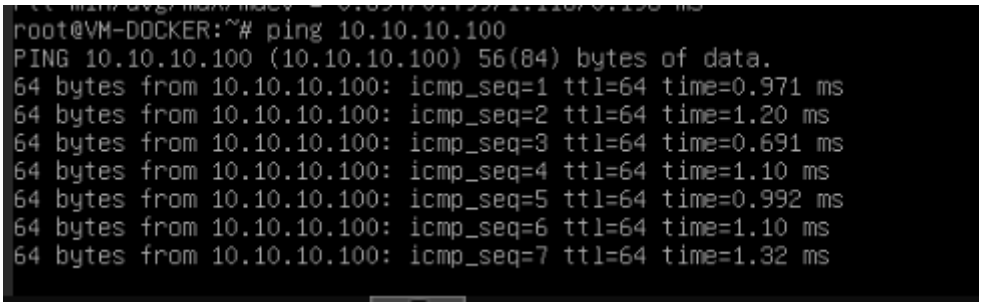
Ping vers VTEP 01



Ping vers VTEP 02 :



Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

1. une requête ARP est envoyée dans le réseau overlay
2. elle est encapsulée dans VXLAN
3. elle est transmise au VTEP distant
4. la machine distante répond
5. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L’analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l’interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d’étendre un réseau de couche 2 au-dessus d’une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d’un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d’étendre un réseau de couche 2 au-dessus d’une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l’illusion d’appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu’une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l’adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l’association entre l’adresse MAC et l’interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu’il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l’architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé
- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

```
L2VPN EVPN Summary
```

On voit :

```
State/PfxRcd   16
```

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

```
MAC
IP
VNI
VTEP source
```

via BGP.

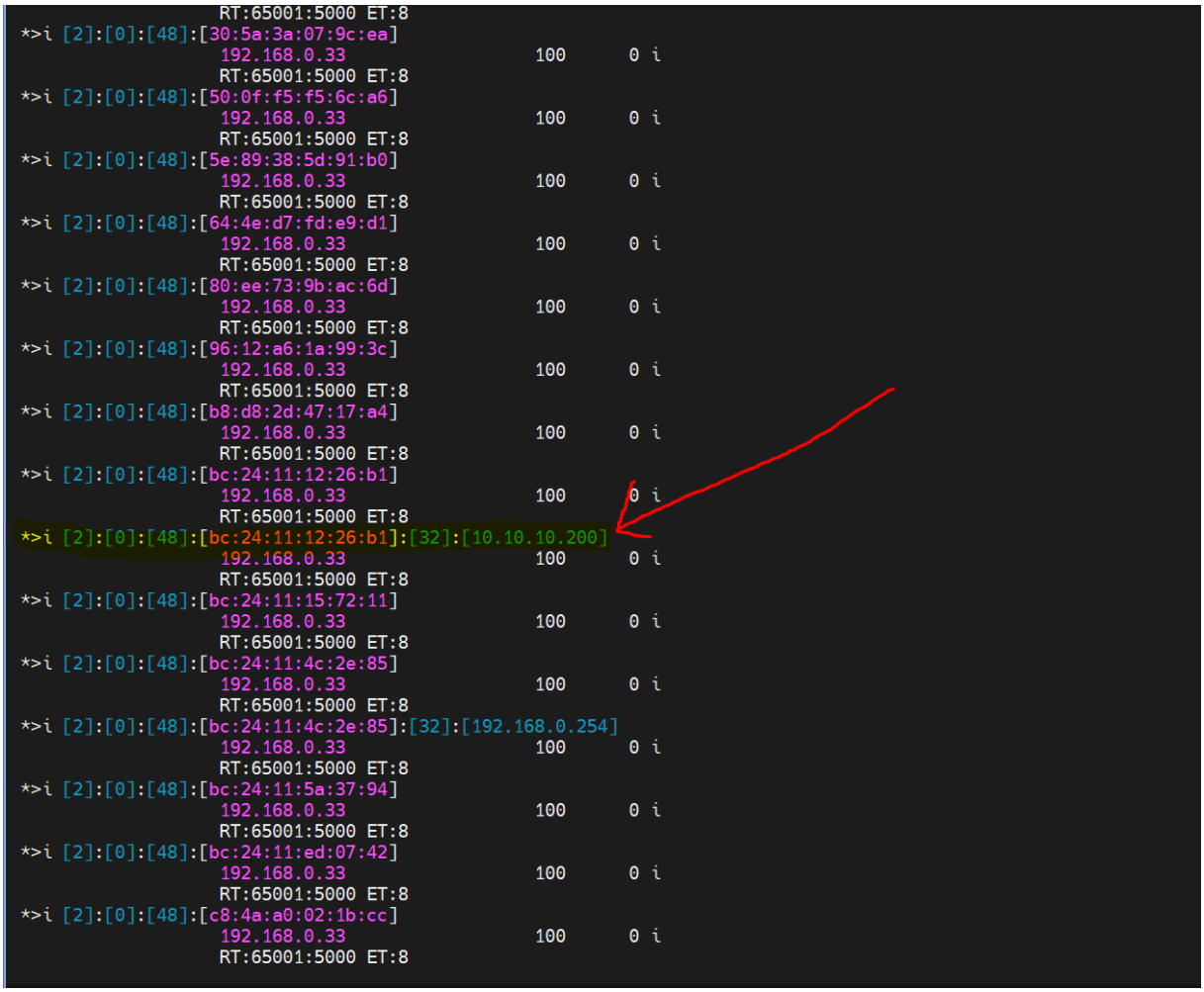
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1

```
grogu# show bgp l2vpn evpn
BGP table version is 7, local router ID is 192.168.0.33
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[EthTag]:[ESI]:[IPlen]:[VTEP-IP]:[Frag-id]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

   Network          Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 10.0.0.139:2
 *>i [2]:[0]:[48]:[bc:24:11:79:43:a4]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:79:43:a4]:[32]:[10.0.0.253]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:c7:fb:75]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:c7:fb:75]:[32]:[10.10.10.100]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:d3:4b:86]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
Route Distinguisher: 192.168.0.33:2
 *>  [2]:[0]:[48]:[00:11:32:c7:10:dc]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
 *>  [2]:[0]:[48]:[10:e9:92:90:54:40]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
 *>  [2]:[0]:[48]:[10:e9:92:90:54:40]:[32]:[192.168.1.1]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
```

Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

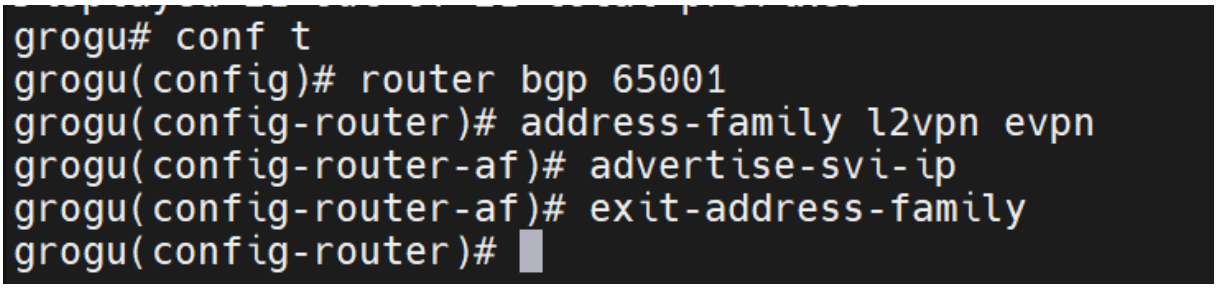
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

6. Une trame Ethernet est créée par une machine dans un VLAN.
7. Cette trame est encapsulée dans un paquet **VXLAN**.
8. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
9. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
10. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

5. Le VTEP reçoit la **trame Ethernet**
6. Il ajoute un **header VXLAN**
7. Il encapsule le tout dans **UDP + IP**
8. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

5. Il retire l'en-tête **IP**
6. Il retire l'en-tête **UDP**
7. Il retire l'en-tête **VXLAN**
8. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

4. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
5. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
6. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :

```
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip link set br0 up
root@grogu:~# ip link set vxlan master br0
Cannot find device "vxlan"
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.2/24 dev br0
root@grogu:~#
```

Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

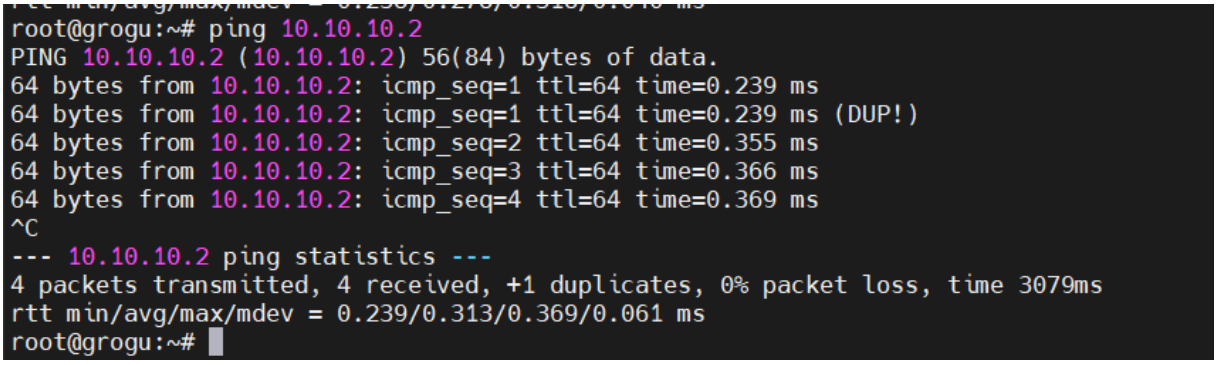
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

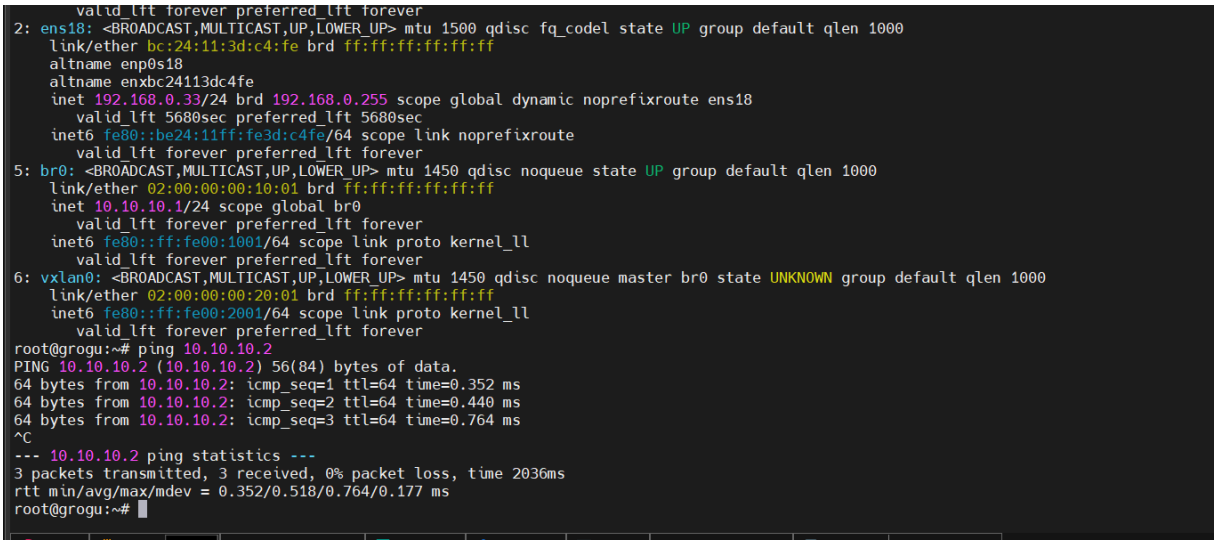


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

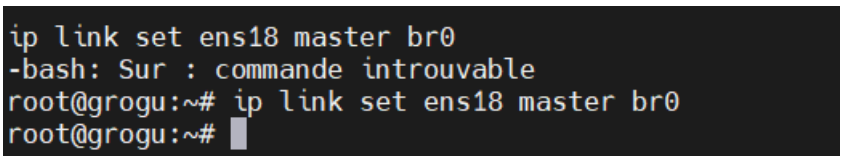
CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

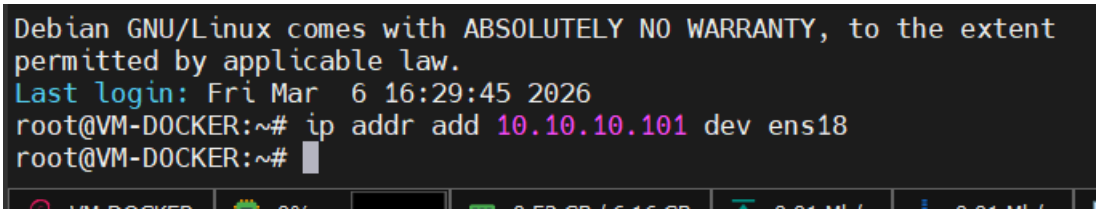
ip link set ens18 master br0



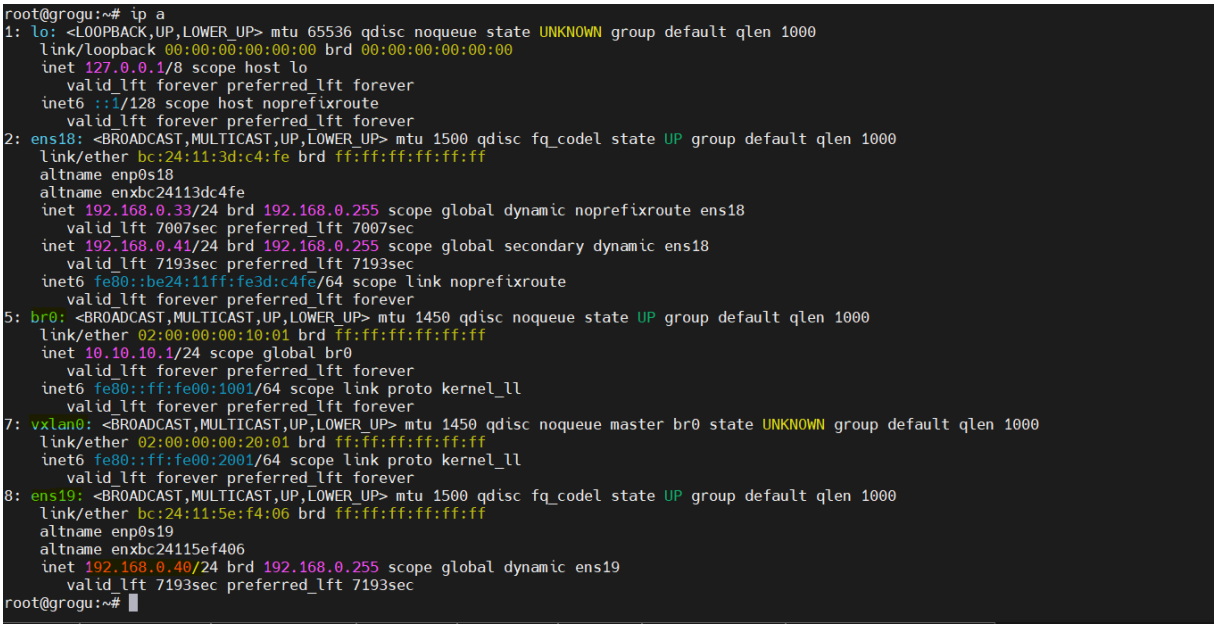
VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse



Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19



Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

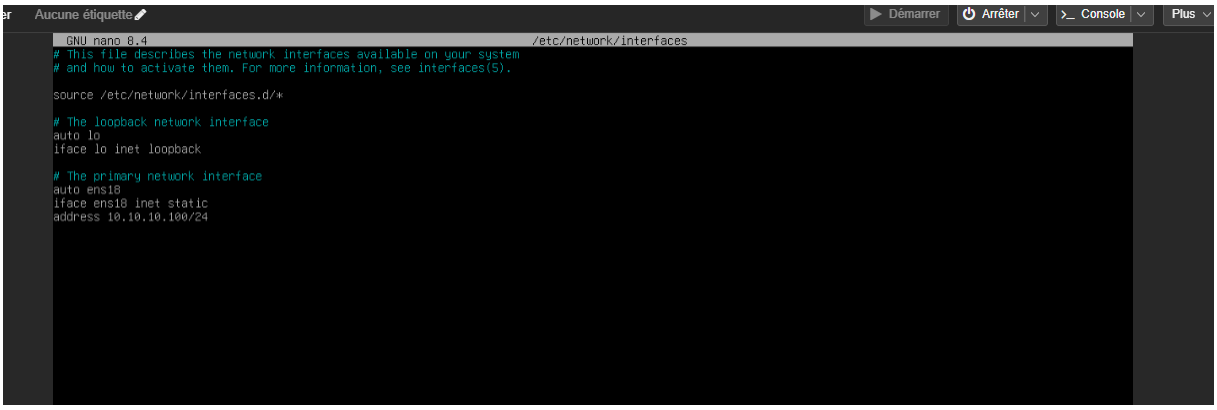
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

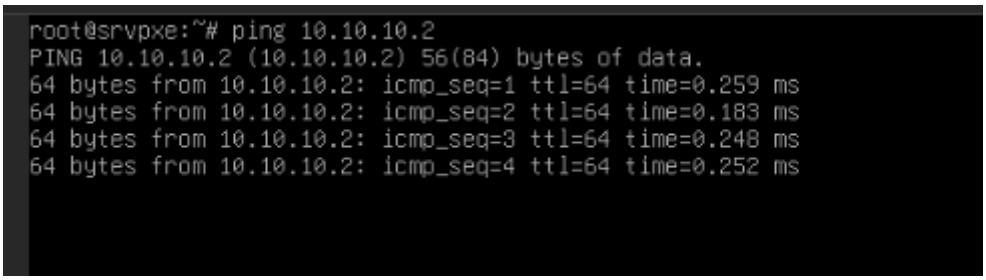
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

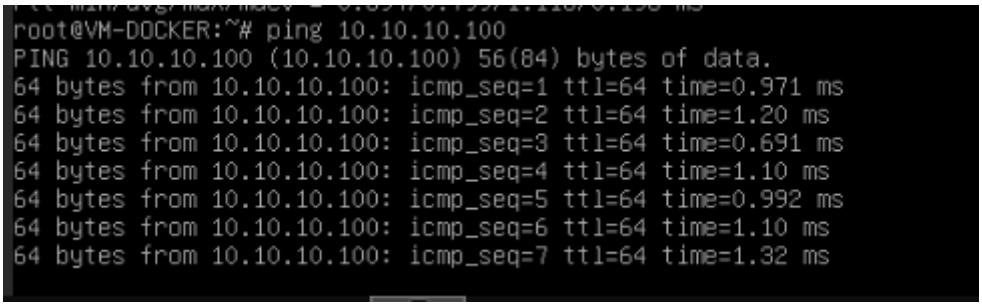
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :

```
root@VM-DOCKER:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.637 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.01 ms
```

Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

6. une requête ARP est envoyée dans le réseau overlay
7. elle est encapsulée dans VXLAN
8. elle est transmise au VTEP distant
9. la machine distante répond
10. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L'analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l'interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d'étendre un réseau de couche 2 au-dessus d'une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d'un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d'étendre un réseau de couche 2 au-dessus d'une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l'illusion d'appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu'une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l'association entre l'adresse MAC et l'interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu'il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l'architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé

- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

`L2VPN EVPN Summary`

On voit :

`State/PfxRcd   16`

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

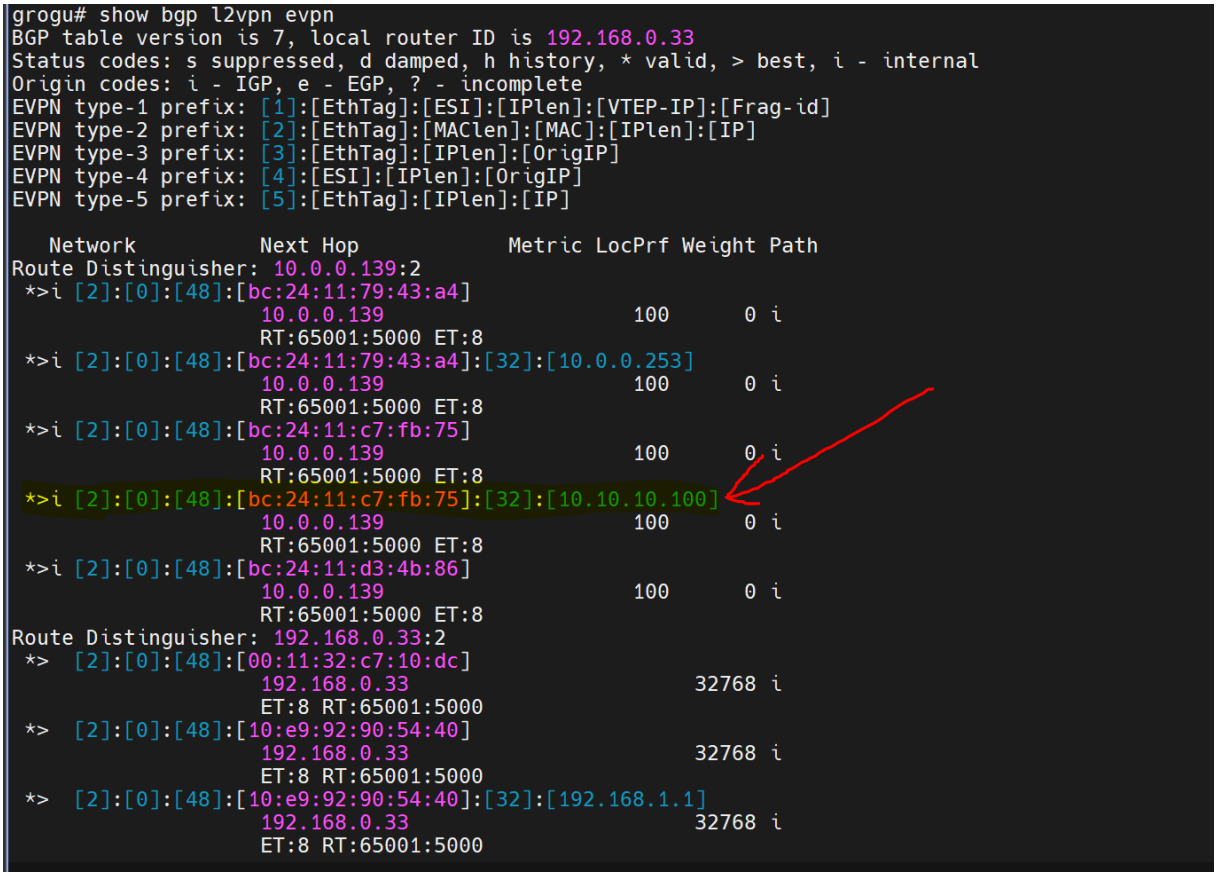
```
MAC
IP
VNI
VTEP source
```

via BGP.

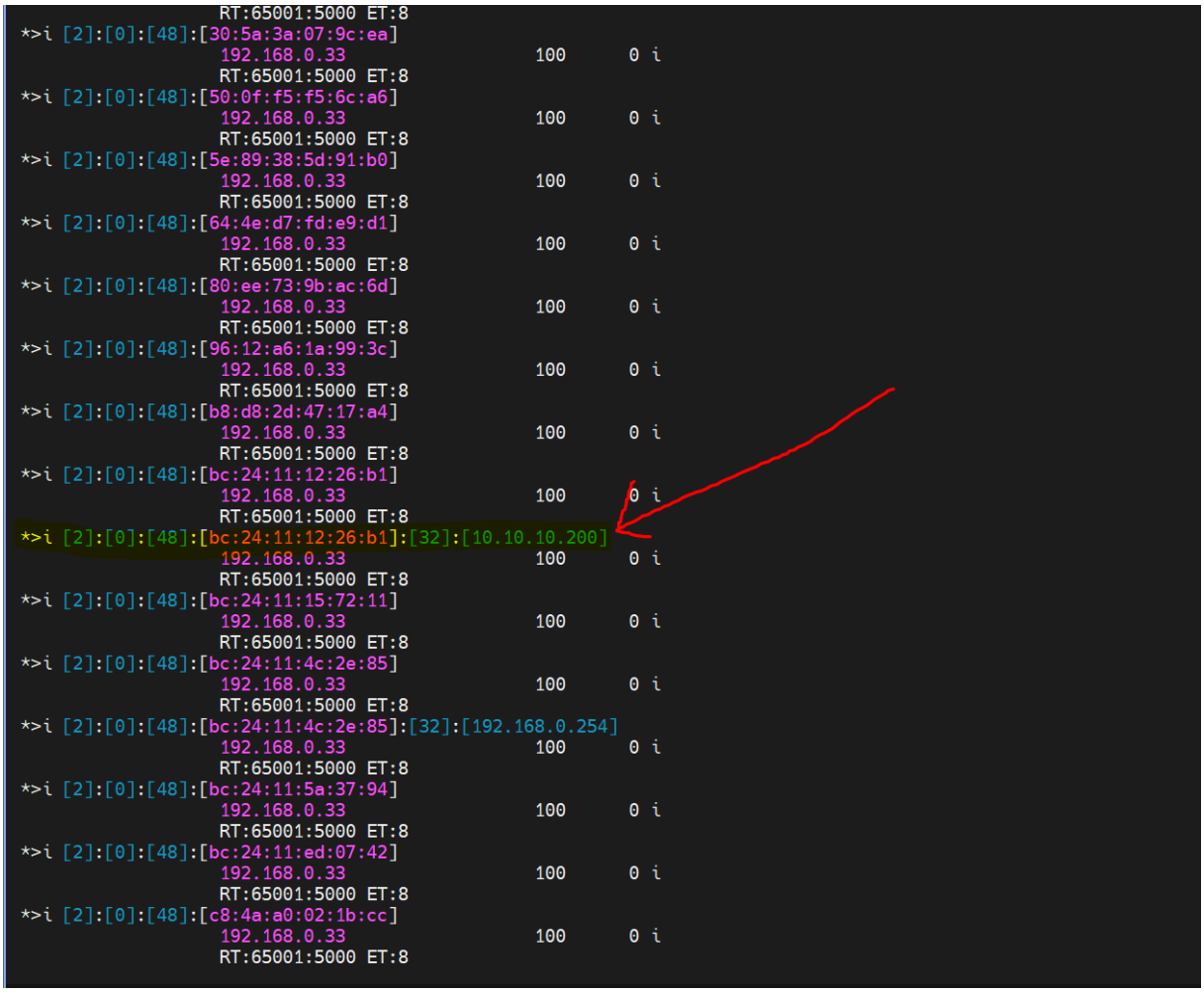
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1



Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

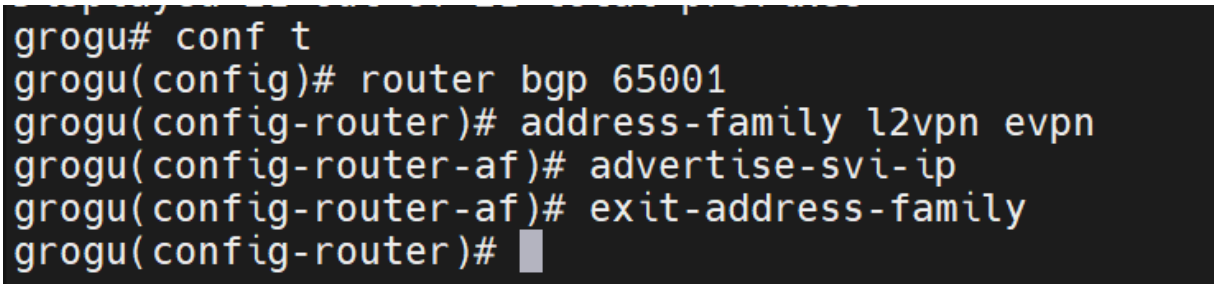
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

11. Une trame Ethernet est créée par une machine dans un VLAN.
12. Cette trame est encapsulée dans un paquet **VXLAN**.
13. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
14. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
15. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

9. Le VTEP reçoit la **trame Ethernet**
10. Il ajoute un **header VXLAN**
11. Il encapsule le tout dans **UDP + IP**
12. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

9. Il retire l'en-tête **IP**
10. Il retire l'en-tête **UDP**
11. Il retire l'en-tête **VXLAN**
12. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

7. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
8. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
9. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :

```
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip link set br0 up
root@grogu:~# ip link set vxlan master br0
Cannot find device "vxlan"
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.2/24 dev br0
root@grogu:~#
```

Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

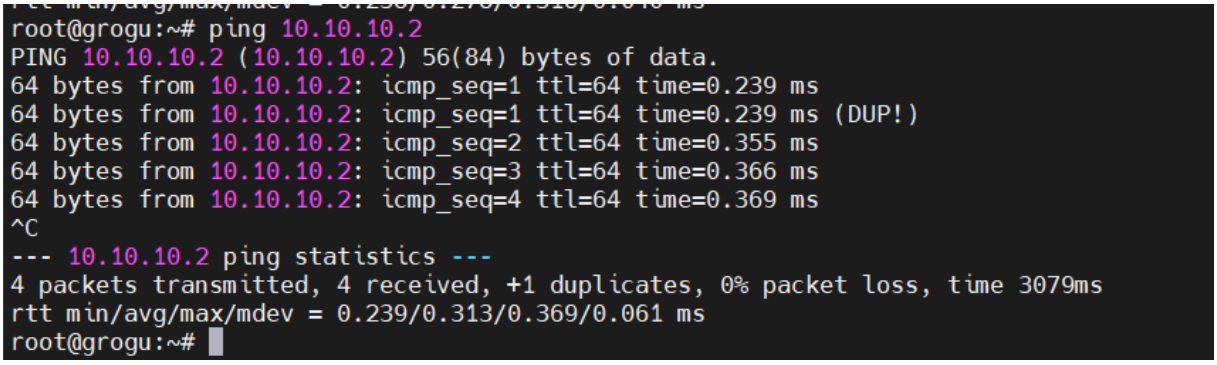
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

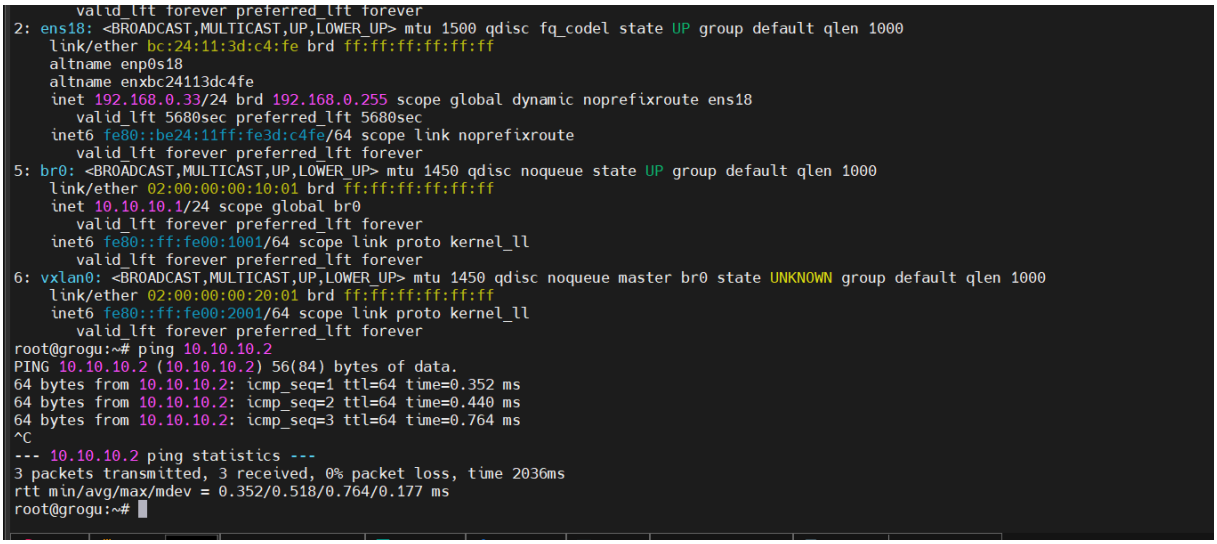


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

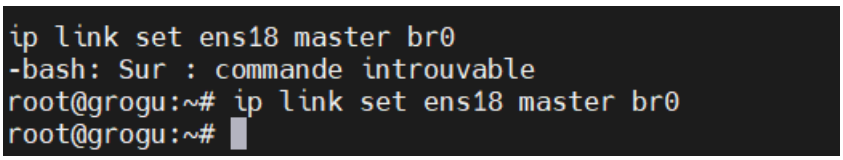
CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

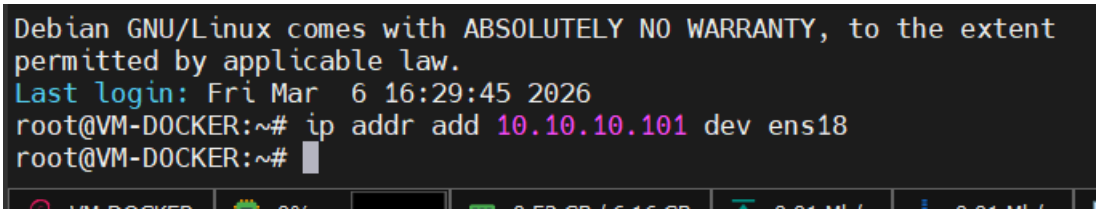
ip link set ens18 master br0



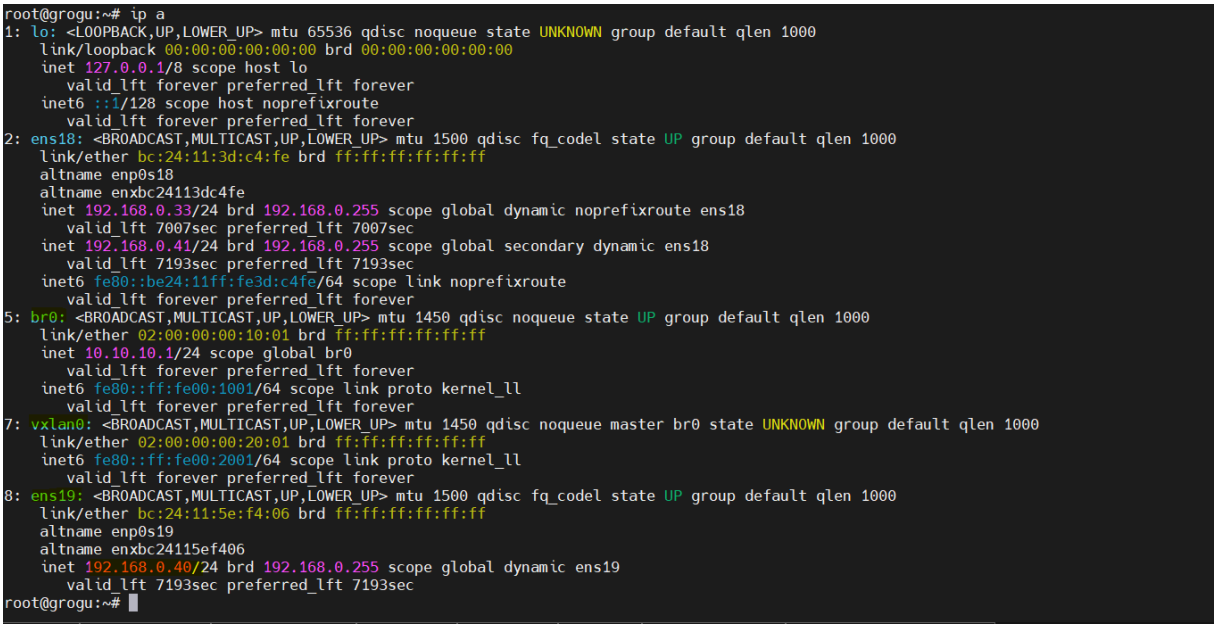
VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse



Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19



Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

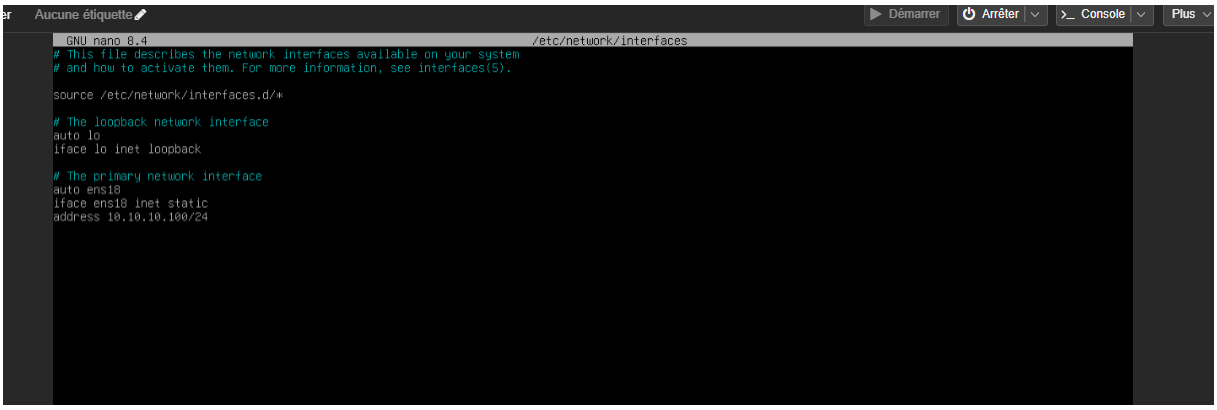
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

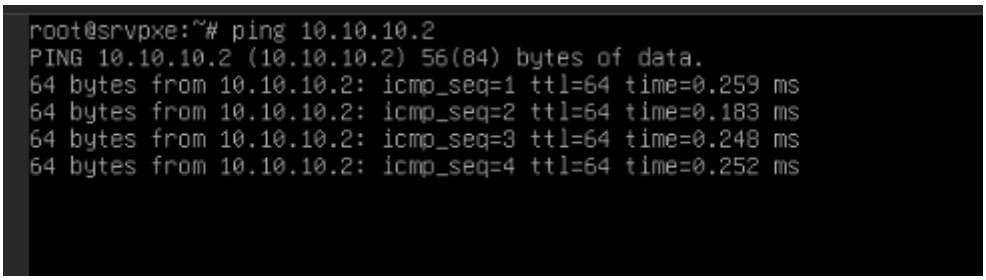
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

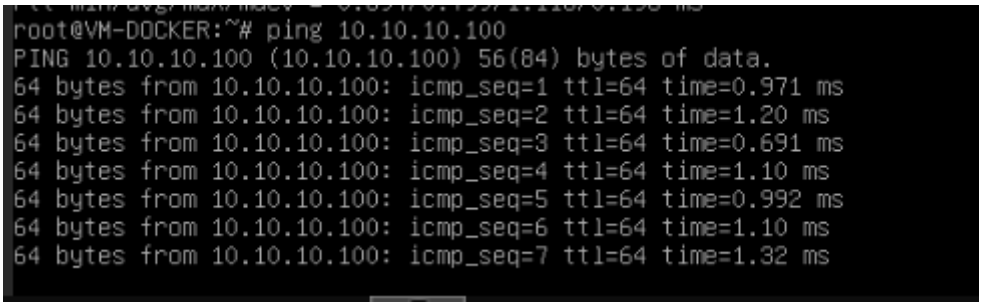
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :

```
root@VM-DOCKER:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.637 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.01 ms
```

Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

11. une requête ARP est envoyée dans le réseau overlay
12. elle est encapsulée dans VXLAN
13. elle est transmise au VTEP distant
14. la machine distante répond
15. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L'analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l'interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d'étendre un réseau de couche 2 au-dessus d'une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d'un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d'étendre un réseau de couche 2 au-dessus d'une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l'illusion d'appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu'une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l'association entre l'adresse MAC et l'interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu'il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l'architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé

- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

`L2VPN EVPN Summary`

On voit :

`State/PfxRcd   16`

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

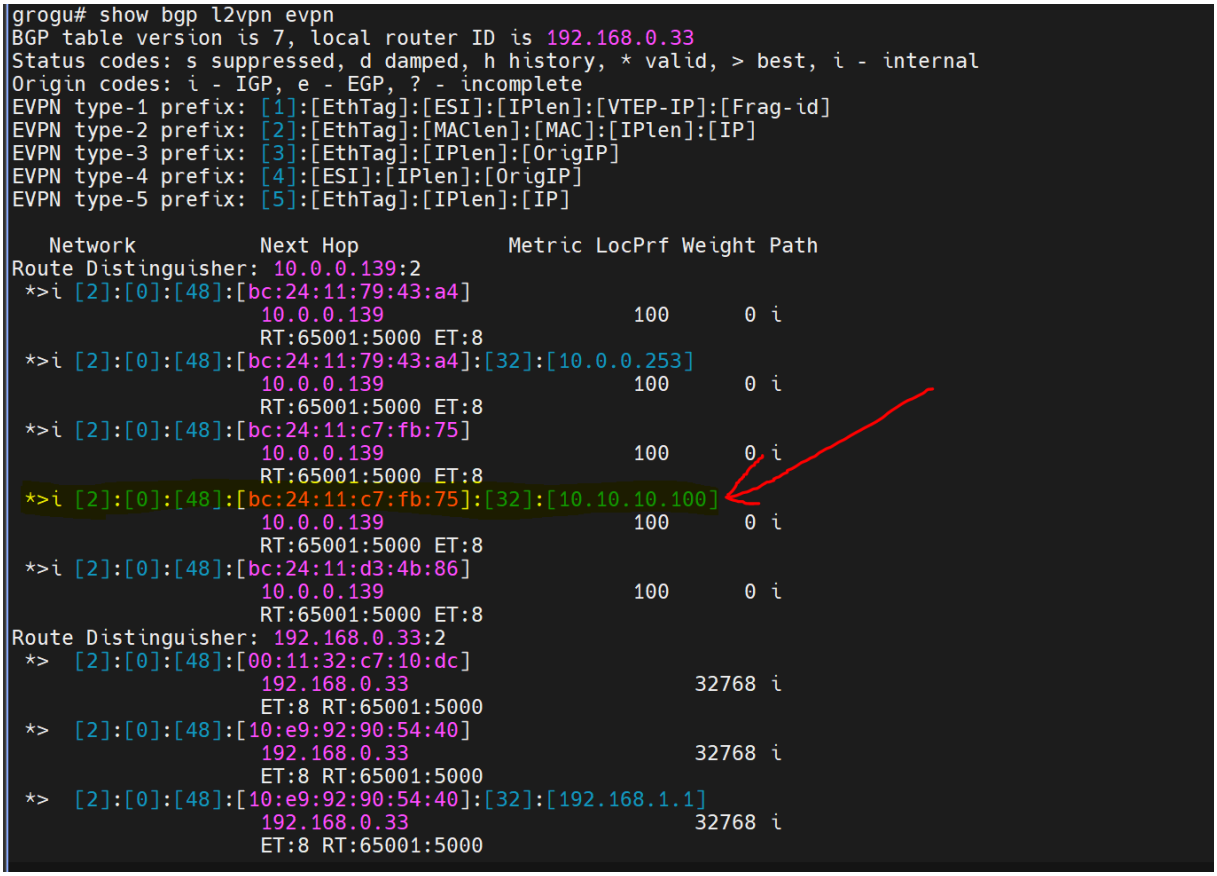
```
MAC
IP
VNI
VTEP source
```

via BGP.

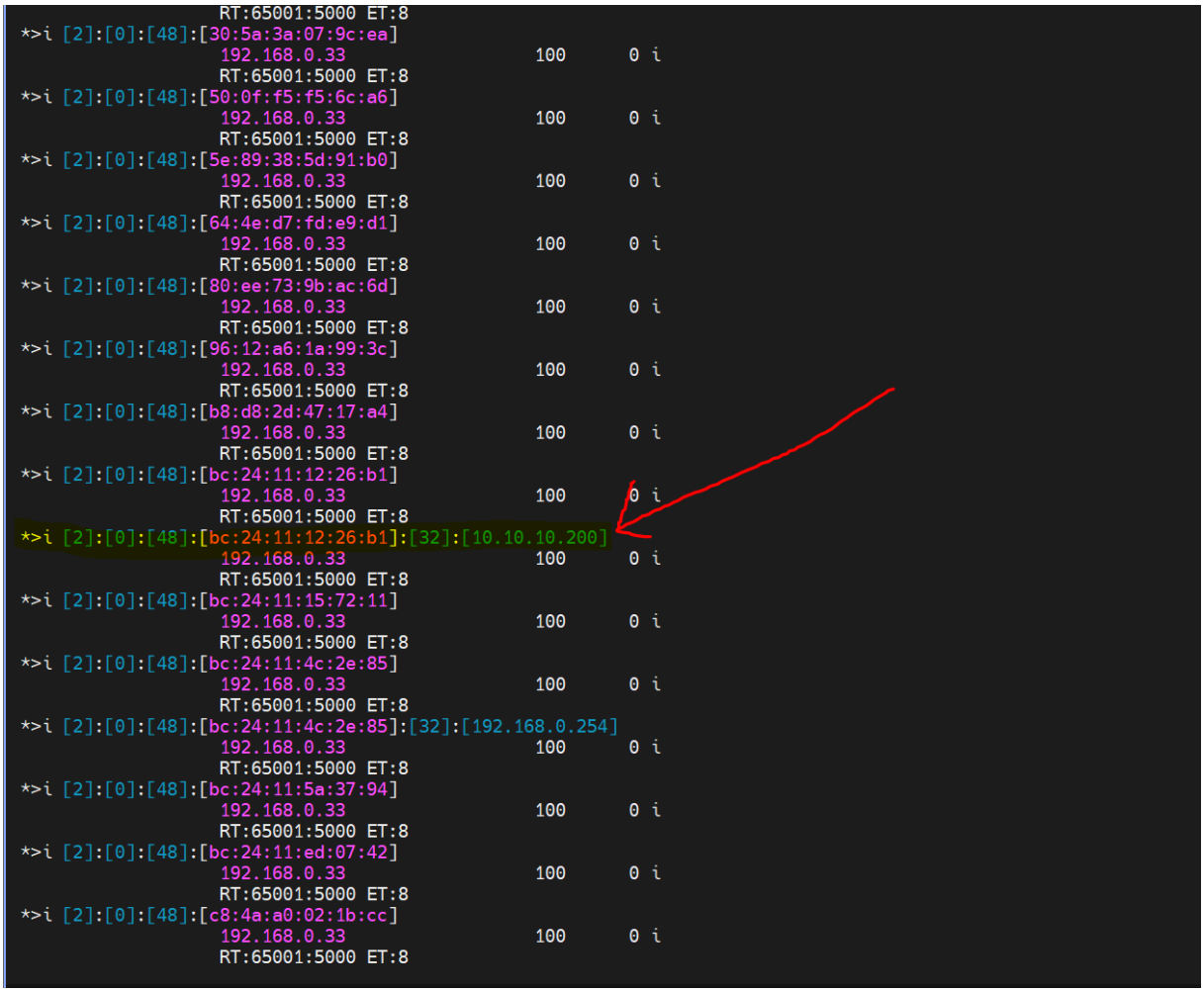
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1



Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

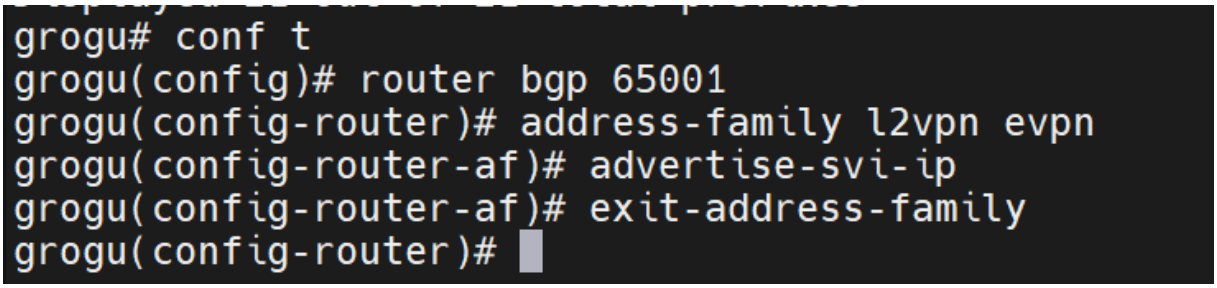
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

16. Une trame Ethernet est créée par une machine dans un VLAN.
17. Cette trame est encapsulée dans un paquet **VXLAN**.
18. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
19. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
20. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

13. Le VTEP reçoit la **trame Ethernet**
14. Il ajoute un **header VXLAN**
15. Il encapsule le tout dans **UDP + IP**
16. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

13. Il retire l'en-tête **IP**
14. Il retire l'en-tête **UDP**
15. Il retire l'en-tête **VXLAN**
16. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

10. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
11. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
12. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :

```
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip link set br0 up
root@grogu:~# ip link set vxlan master br0
Cannot find device "vxlan"
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.2/24 dev br0
root@grogu:~#
```

Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

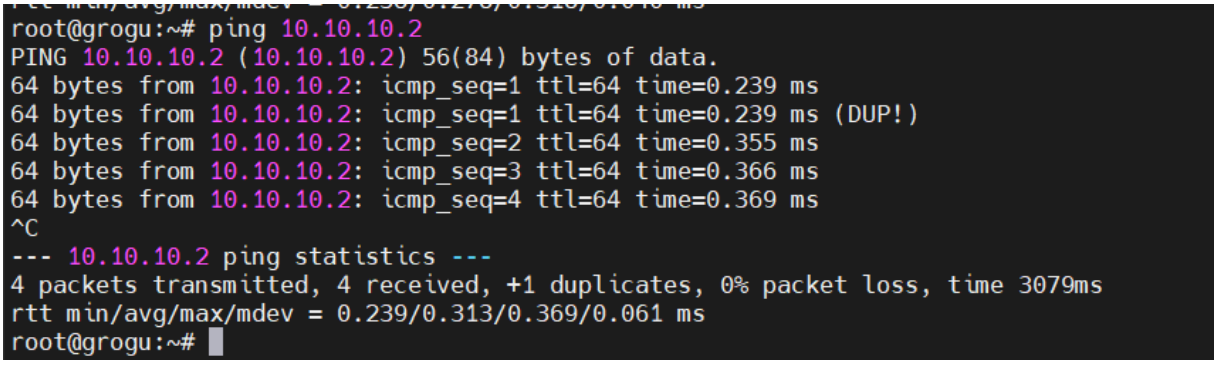
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

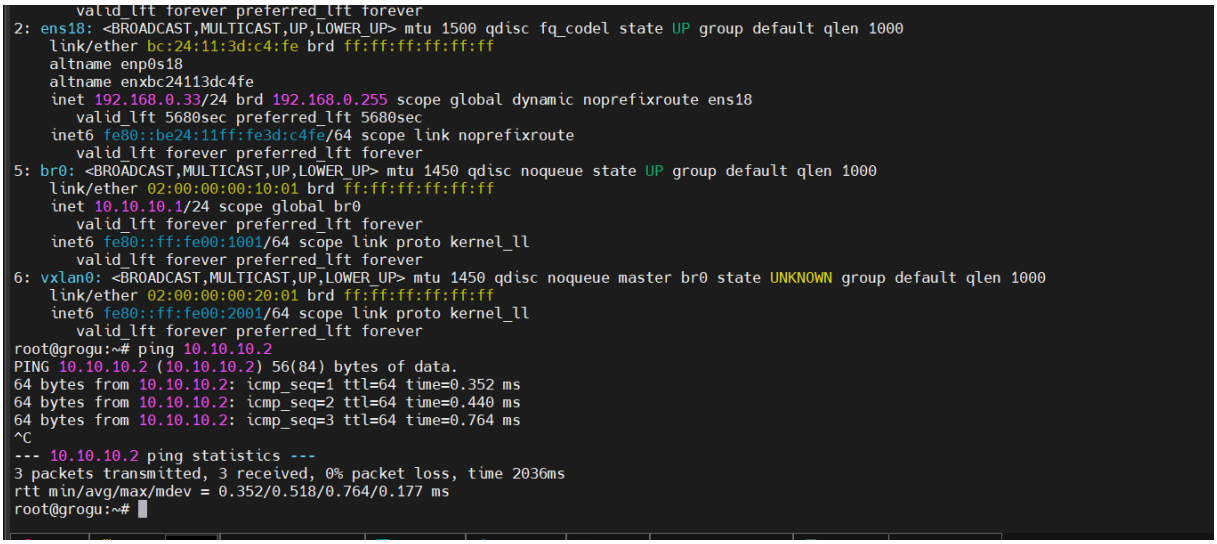


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

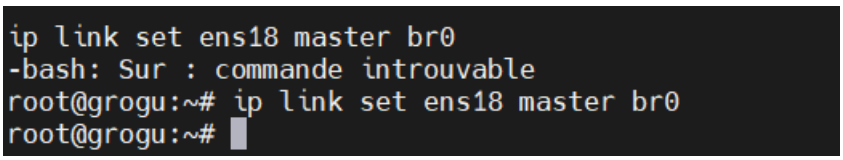
CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

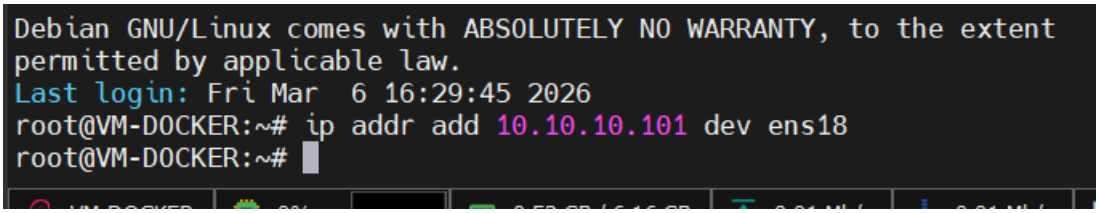
ip link set ens18 master br0



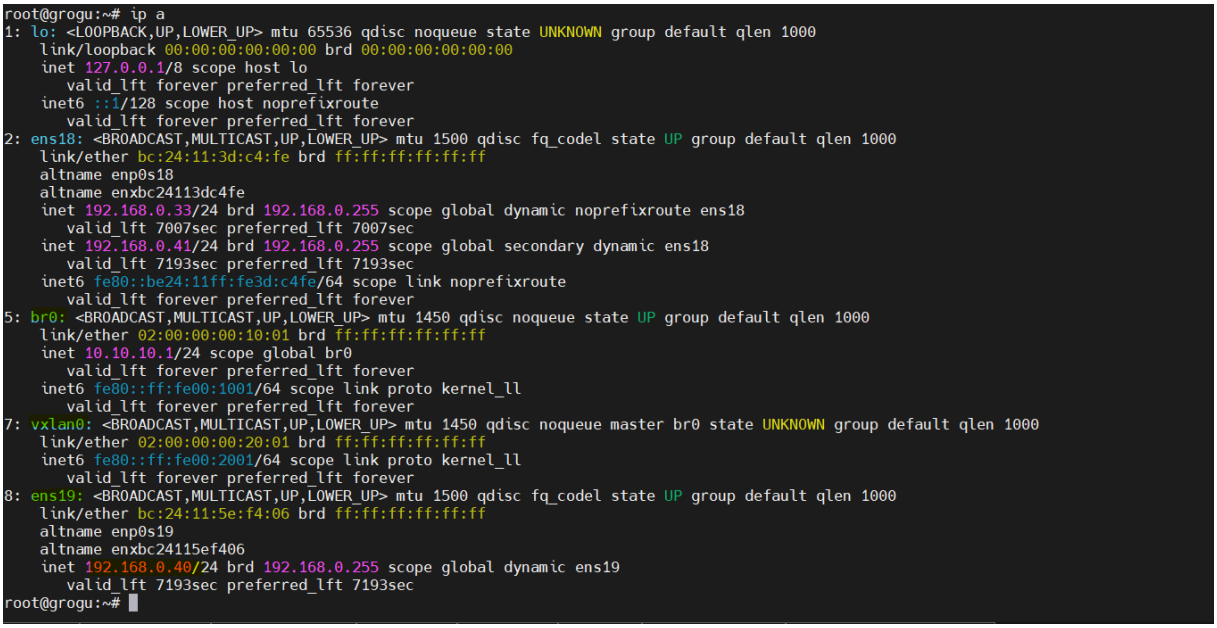
VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse



Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19



Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

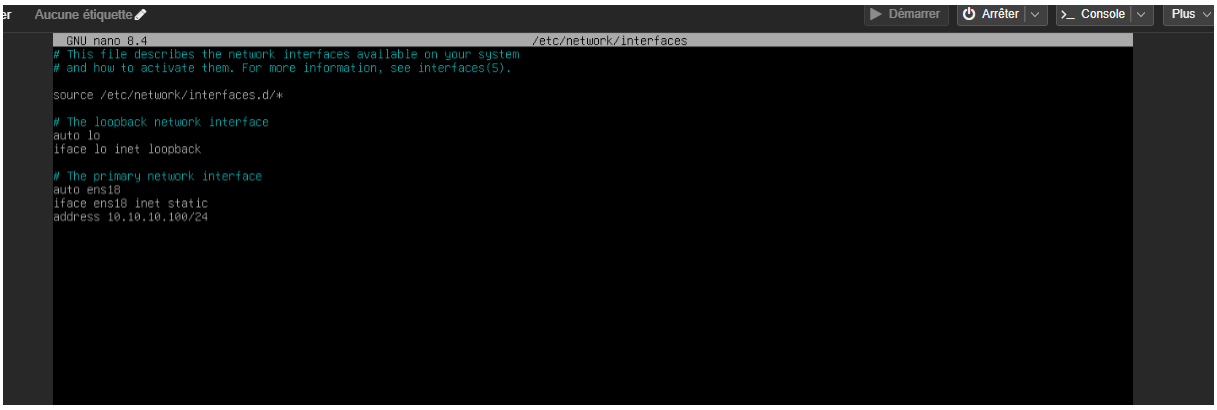
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

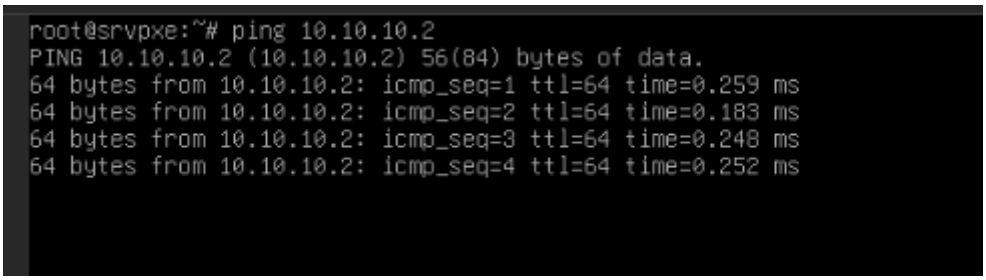
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

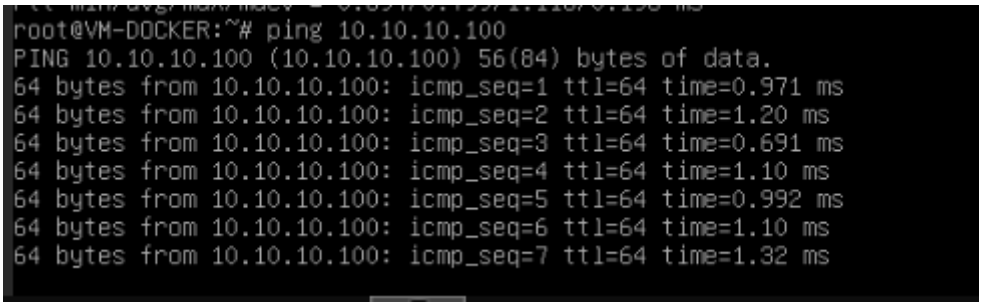
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :

```
root@VM-DOCKER:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.637 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.01 ms
```

Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

16. une requête ARP est envoyée dans le réseau overlay
17. elle est encapsulée dans VXLAN
18. elle est transmise au VTEP distant
19. la machine distante répond
20. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L'analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l'interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d'étendre un réseau de couche 2 au-dessus d'une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d'un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d'étendre un réseau de couche 2 au-dessus d'une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l'illusion d'appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu'une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l'association entre l'adresse MAC et l'interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu'il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l'architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé

- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

`L2VPN EVPN Summary`

On voit :

`State/PfxRcd   16`

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

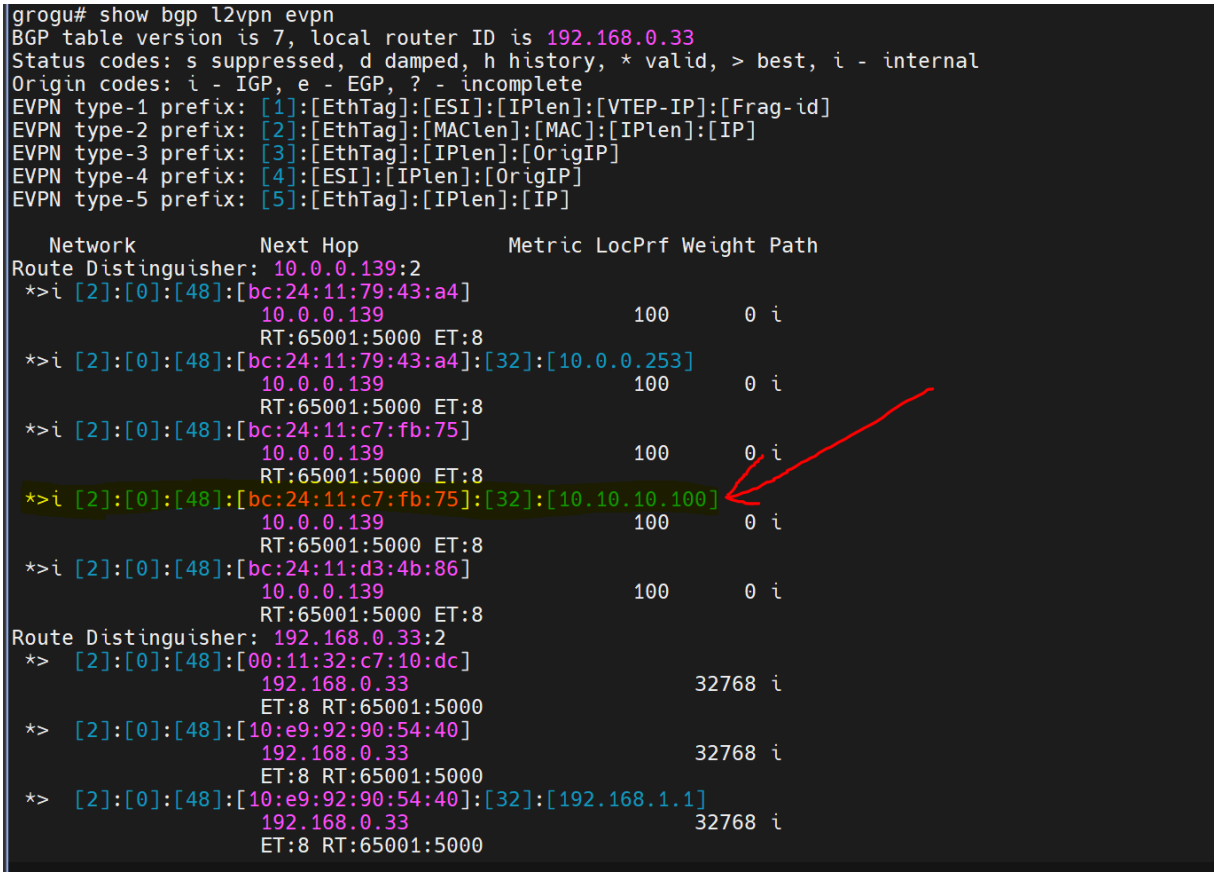
```
MAC
IP
VNI
VTEP source
```

via BGP.

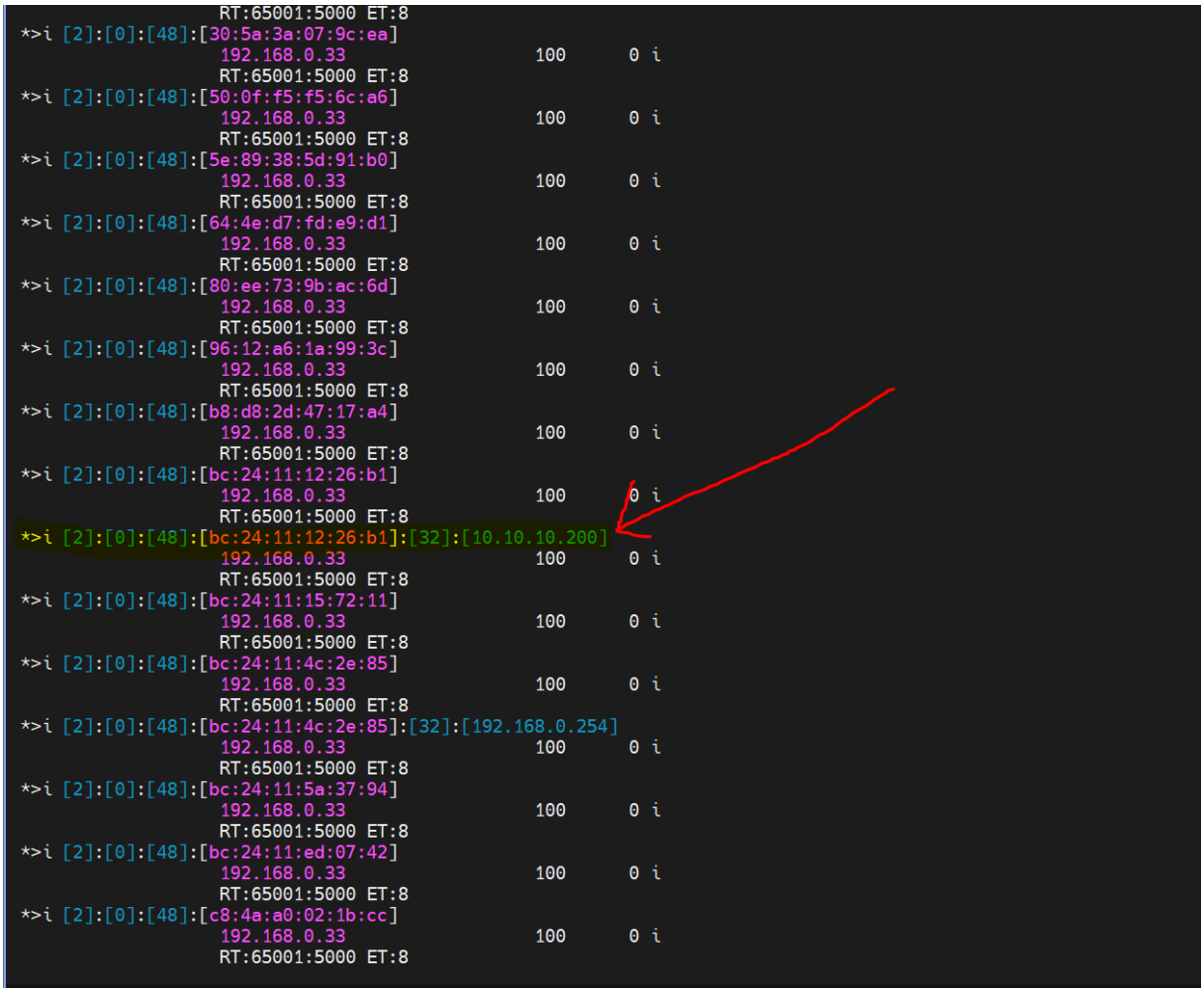
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1



Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

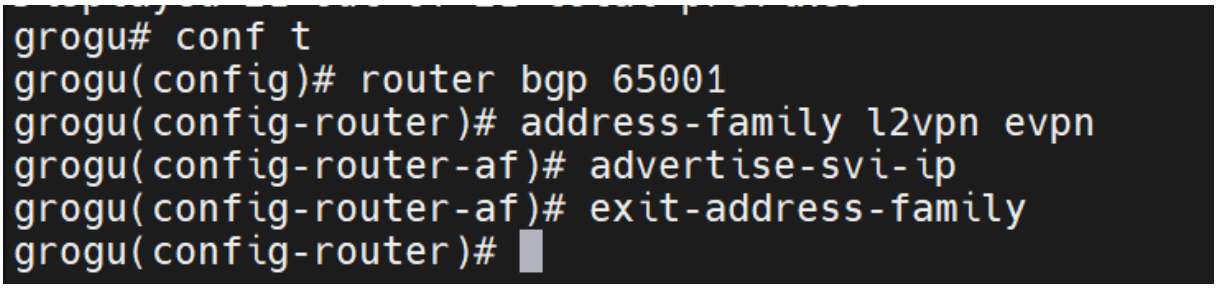
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

bb

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

21. Une trame Ethernet est créée par une machine dans un VLAN.
22. Cette trame est encapsulée dans un paquet **VXLAN**.
23. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
24. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
25. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**

- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

17. Le VTEP reçoit la **trame Ethernet**
18. Il ajoute un **header VXLAN**
19. Il encapsule le tout dans **UDP + IP**
20. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

17. Il retire l'en-tête **IP**
18. Il retire l'en-tête **UDP**
19. Il retire l'en-tête **VXLAN**
20. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

13. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
14. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
15. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

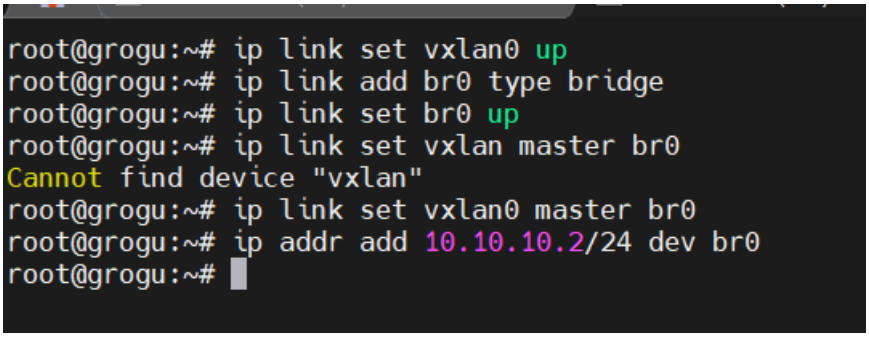
ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :



Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite

été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1

Machine	IP Underlay	IP Overlay
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

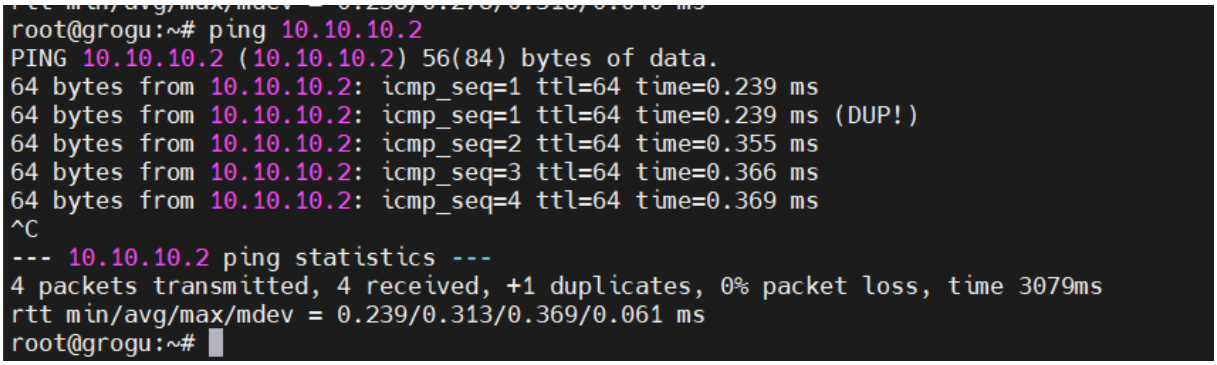
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

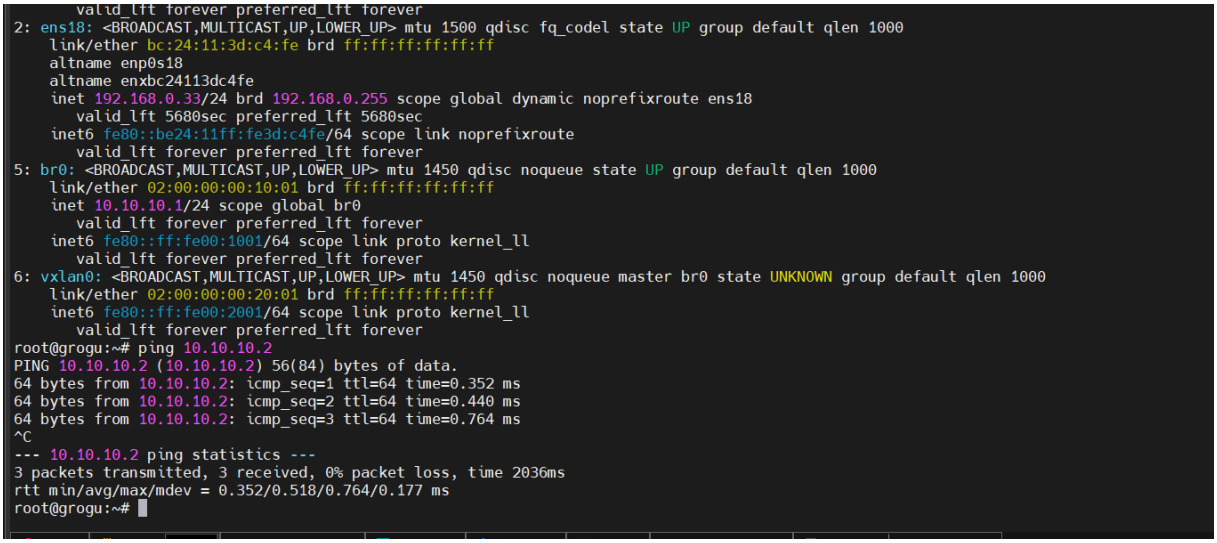


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

ip link set ens18 master br0

```
ip link set ens18 master br0
-bash: Sur : commande introuvable
root@grogu:~# ip link set ens18 master br0
root@grogu:~#
```

VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Mar  6 16:29:45 2026
root@VM-DOCKER:~# ip addr add 10.10.10.101 dev ens18
root@VM-DOCKER:~#
```

Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19

```
root@grogu:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:3d:c4:fe brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc24113dc4fe
    inet 192.168.0.33/24 brd 192.168.0.255 scope global dynamic noprefixroute ens18
       valid_lft 7007sec preferred_lft 7007sec
    inet 192.168.0.41/24 brd 192.168.0.255 scope global secondary dynamic ens18
       valid_lft 7193sec preferred_lft 7193sec
    inet6 fe80::be24:11ff:fe3d:c4fe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
5: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    link/ether 02:00:00:00:10:01 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.1/24 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::ff:fe00:1001/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br0 state UNKNOWN group default qlen 1000
    link/ether 02:00:00:00:20:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ff:fe00:2001/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:5e:f4:06 brd ff:ff:ff:ff:ff:ff
    altname enp0s19
    altname enxbc24115ef406
    inet 192.168.0.40/24 brd 192.168.0.255 scope global dynamic ens19
       valid_lft 7193sec preferred_lft 7193sec
root@grogu:~#
```

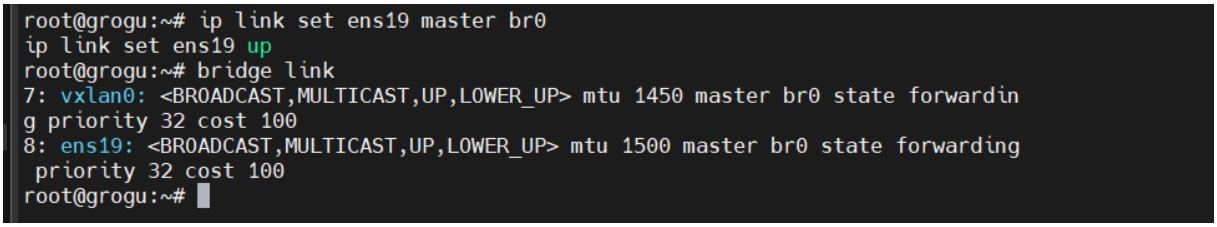
Comme on le voit ici j’ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

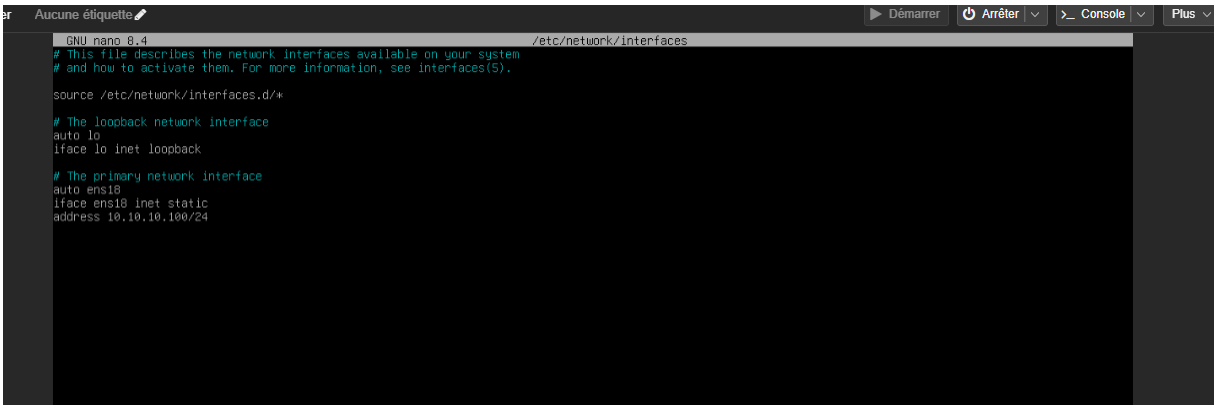
bridge link



Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000 0000       ..............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000 0000       ..............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès

```
root@srvpxe:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.259 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.183 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=0.248 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=0.252 ms
```

Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

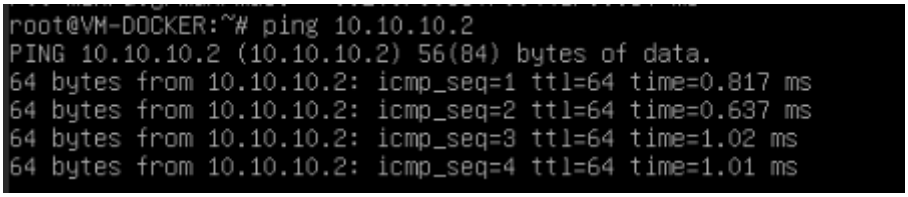
Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

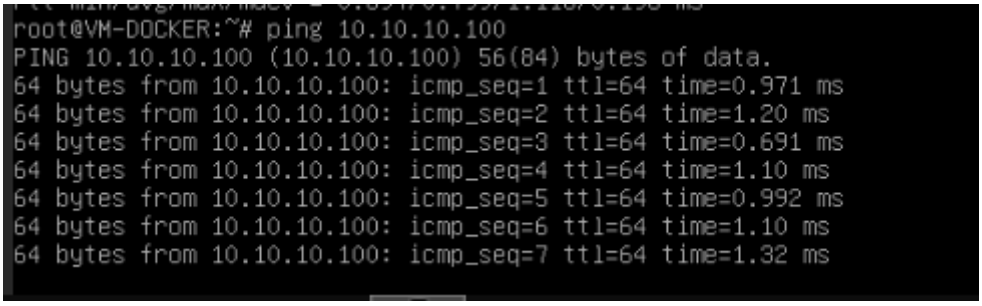
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :



Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

21. une requête ARP est envoyée dans le réseau overlay
22. elle est encapsulée dans VXLAN
23. elle est transmise au VTEP distant
24. la machine distante répond
25. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L’analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l’interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d’étendre un réseau de couche 2 au-dessus d’une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d’un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d’étendre un réseau de couche 2 au-dessus d’une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l’illusion d’appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu’une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l’adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l’association entre l’adresse MAC et l’interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu’il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l’architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé
- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

```
L2VPN EVPN Summary
```

On voit :

```
State/PfxRcd   16
```

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

```
MAC
IP
VNI
VTEP source
```

via BGP.

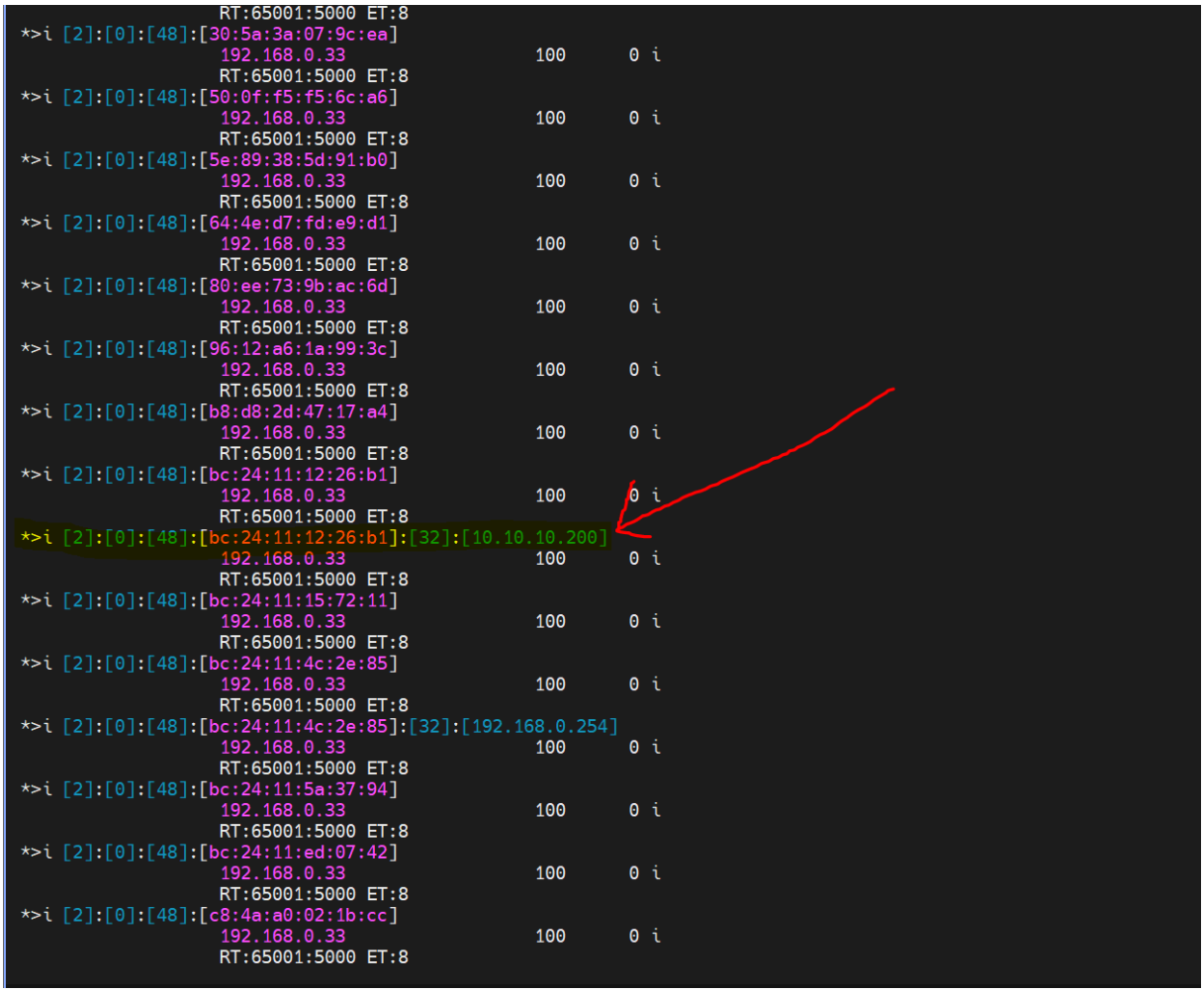
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1

```
grogu# show bgp l2vpn evpn
BGP table version is 7, local router ID is 192.168.0.33
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[EthTag]:[ESI]:[IPlen]:[VTEP-IP]:[Frag-id]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

   Network          Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 10.0.0.139:2
 *>i [2]:[0]:[48]:[bc:24:11:79:43:a4]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:79:43:a4]:[32]:[10.0.0.253]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:c7:fb:75]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:c7:fb:75]:[32]:[10.10.10.100]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
 *>i [2]:[0]:[48]:[bc:24:11:d3:4b:86]
                    10.0.0.139                    100      0 i
                    RT:65001:5000 ET:8
Route Distinguisher: 192.168.0.33:2
 *>  [2]:[0]:[48]:[00:11:32:c7:10:dc]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
 *>  [2]:[0]:[48]:[10:e9:92:90:54:40]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
 *>  [2]:[0]:[48]:[10:e9:92:90:54:40]:[32]:[192.168.1.1]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000
```

Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

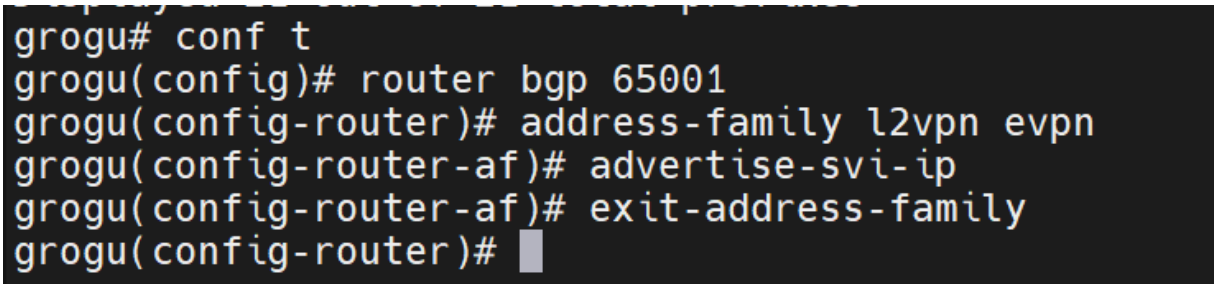
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

26. Une trame Ethernet est créée par une machine dans un VLAN.
27. Cette trame est encapsulée dans un paquet **VXLAN**.
28. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
29. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
30. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

1. Le VTEP reçoit la **trame Ethernet**
2. Il ajoute un **header VXLAN**
3. Il encapsule le tout dans **UDP + IP**
4. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

1. Il retire l'en-tête **IP**
2. Il retire l'en-tête **UDP**
3. Il retire l'en-tête **VXLAN**
4. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

16. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
17. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
18. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :

```
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip link set br0 up
root@grogu:~# ip link set vxlan master br0
Cannot find device "vxlan"
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.2/24 dev br0
root@grogu:~#
```

Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

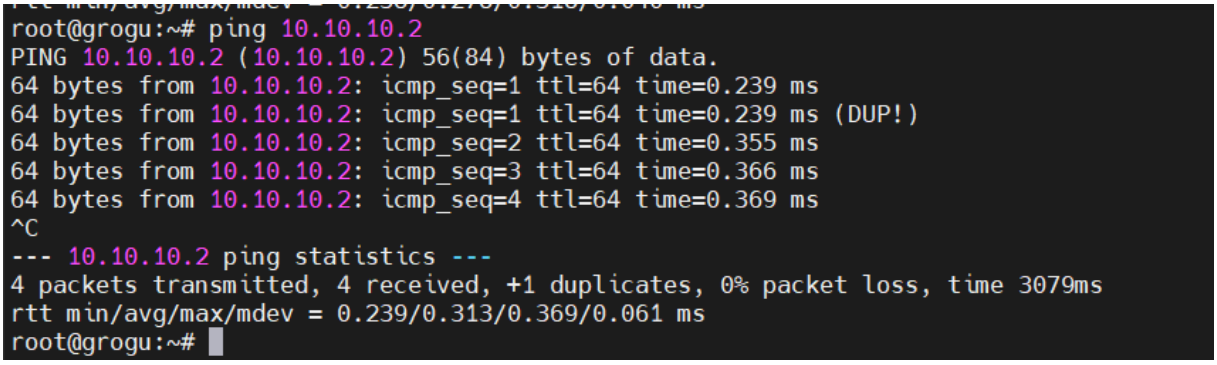
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

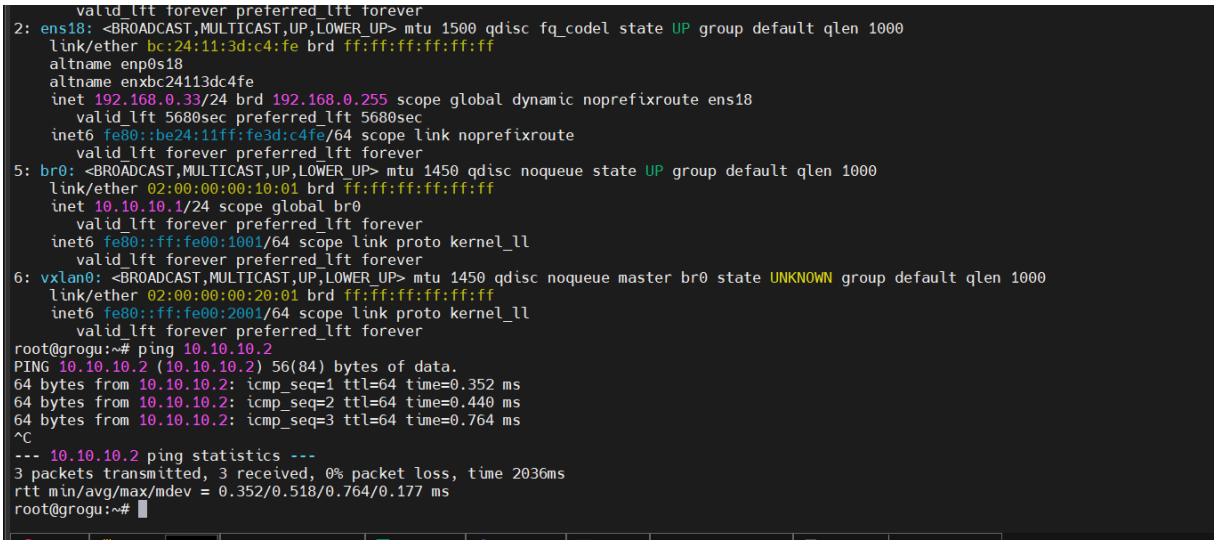


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

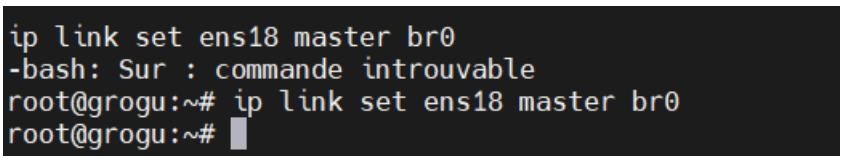
CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

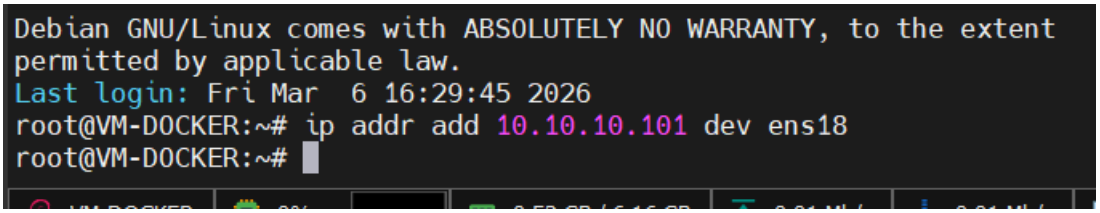
ip link set ens18 master br0



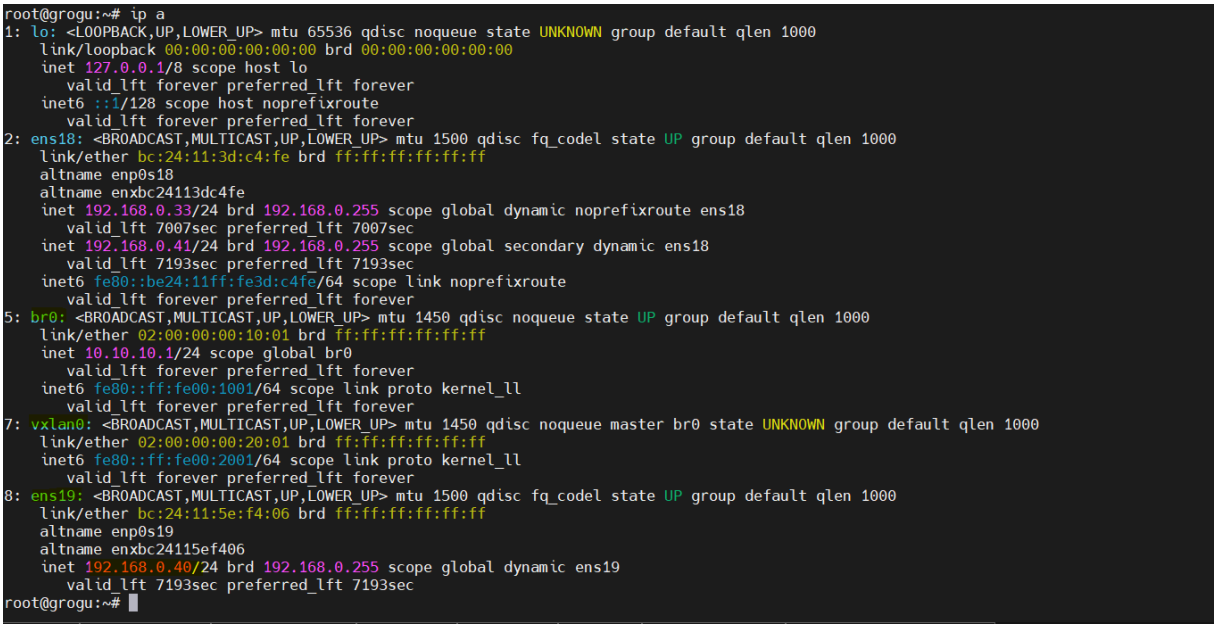
VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse



Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19



Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

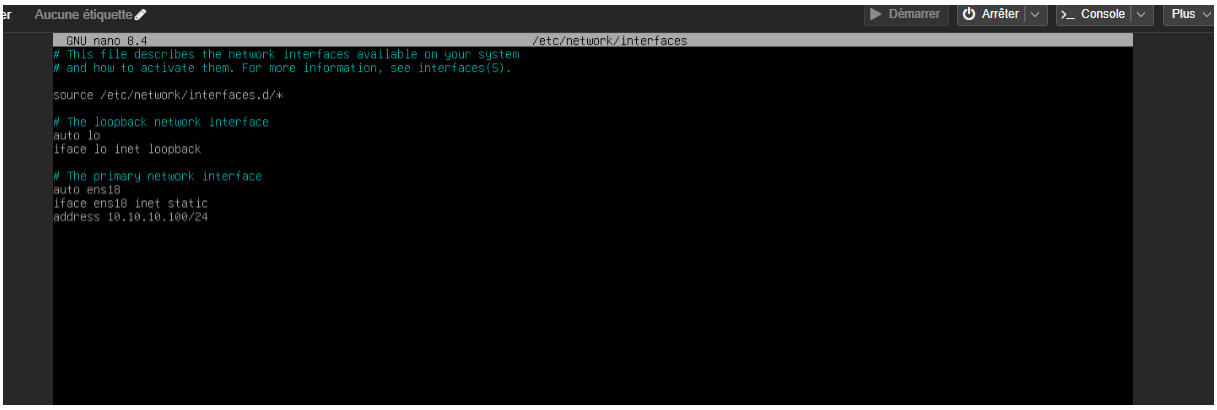
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

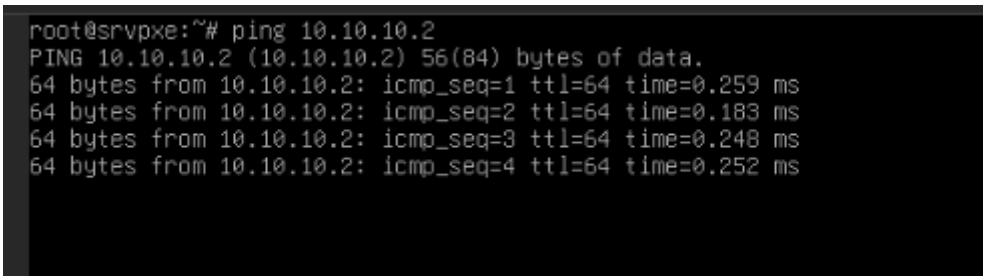
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

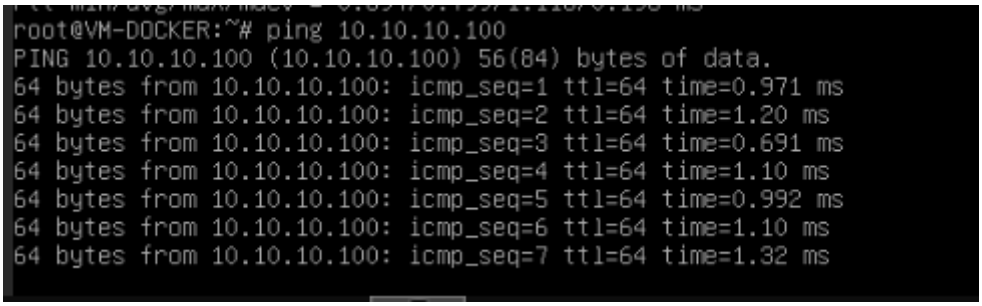
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :

```
root@VM-DOCKER:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.637 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.01 ms
```

Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

26. une requête ARP est envoyée dans le réseau overlay
27. elle est encapsulée dans VXLAN
28. elle est transmise au VTEP distant
29. la machine distante répond
30. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L'analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l'interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d'étendre un réseau de couche 2 au-dessus d'une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d'un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d'étendre un réseau de couche 2 au-dessus d'une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l'illusion d'appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu'une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l'association entre l'adresse MAC et l'interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu'il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l'architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé

- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

`L2VPN EVPN Summary`

On voit :

`State/PfxRcd   16`

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

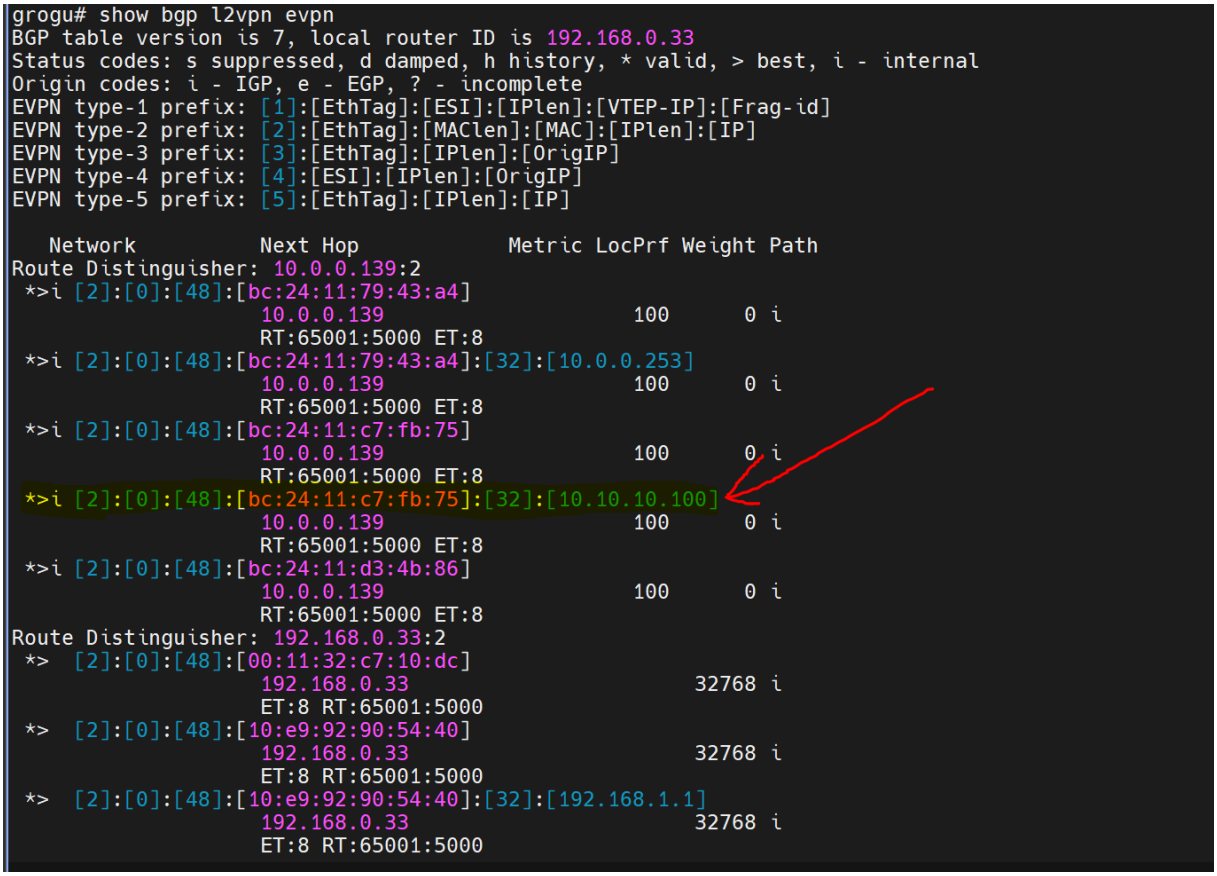
```
MAC
IP
VNI
VTEP source
```

via BGP.

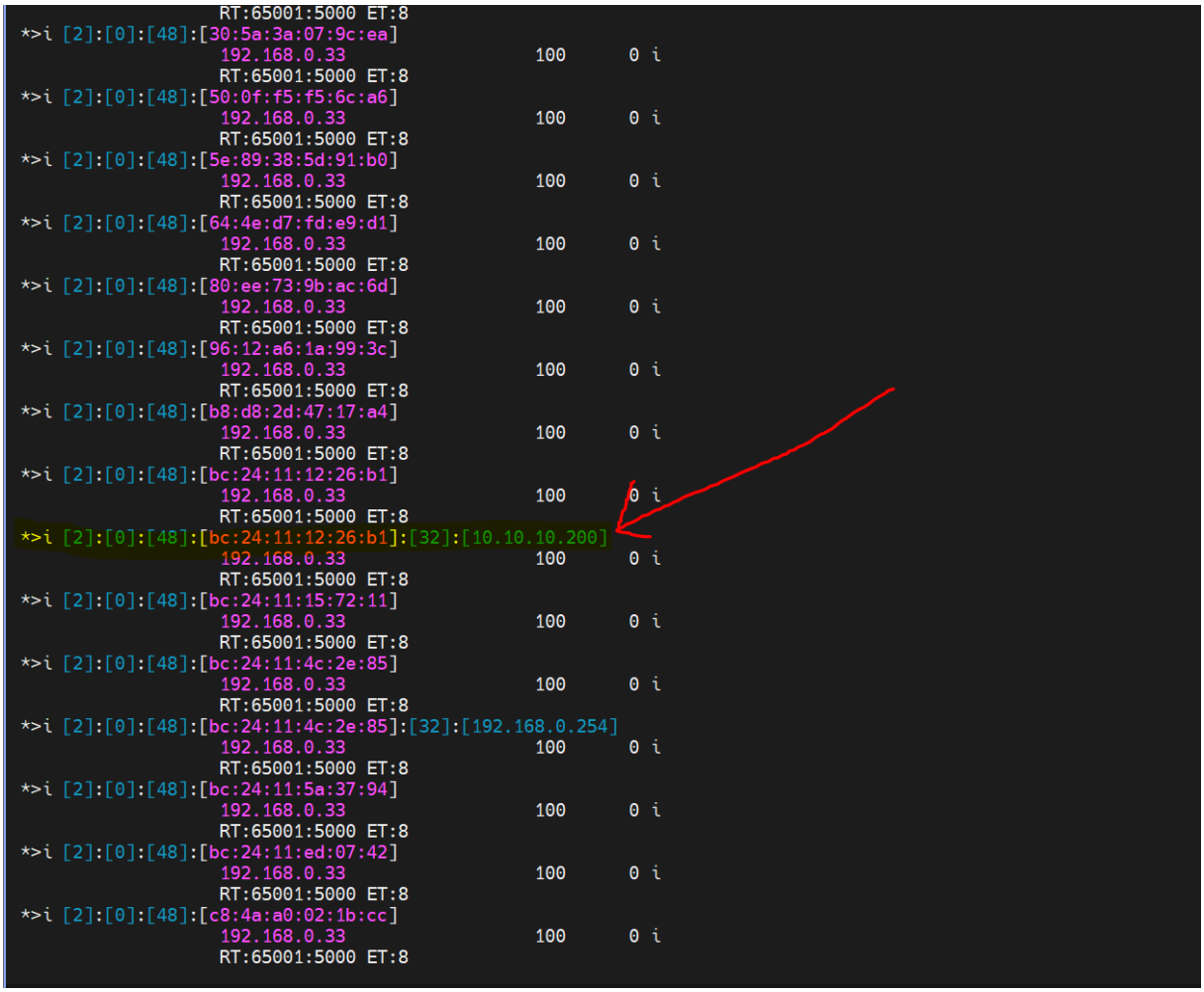
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1



Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

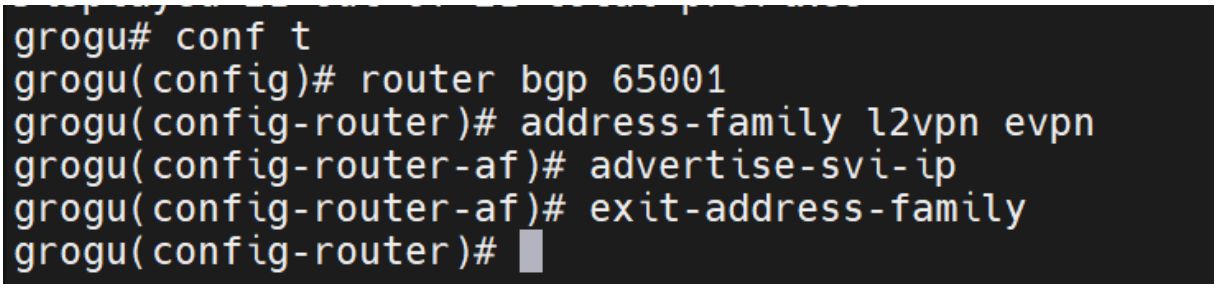
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place VXLAN

VXLAN (Virtual Extensible LAN) est une technologie permettant d'étendre un réseau de niveau 2 (VLAN) à travers un réseau de niveau 3.

Elle permet de **transporter des VLAN entre différents sites ou hôtes** en encapsulant les trames Ethernet dans des **paquets UDP**, afin de les faire circuler sur un réseau IP.

Ainsi, des machines situées sur des réseaux physiques différents peuvent **appartenir au même réseau logique de couche 2**.

Principe

Le fonctionnement de VXLAN repose sur l'encapsulation :

31. Une trame Ethernet est créée par une machine dans un VLAN.
32. Cette trame est encapsulée dans un paquet **VXLAN**.
33. Le paquet VXLAN est ensuite encapsulé dans **UDP/IP**.
34. Le réseau IP transporte ce paquet jusqu'à l'autre extrémité.
35. Le paquet est décapsulé et la trame Ethernet originale est restituée.

Objectifs principaux de VXLAN

- Étendre des réseaux **Layer 2 sur des infrastructures Layer 3**
- Permettre l'interconnexion de **datacenters ou serveurs distants**
- Dépasser la limite des **4096 VLAN classiques**
- Faciliter les architectures **cloud et virtualisation**

Principe du VTEP

Un **VTEP (VXLAN Tunnel End Point)** est l'équipement qui permet de **faire la conversion entre le réseau VLAN (couche 2) et le réseau VXLAN (couche 3)**.

Autrement dit, le VTEP est **le point d'entrée et de sortie du tunnel VXLAN**.

Il peut être :

- un **switch physique**
- un **hyperviseur**
- un **serveur Linux**
- un **routeur**

Fonctionnement

Le rôle du VTEP est double :

1️⃣ Encapsulation

Lorsqu'une trame Ethernet arrive depuis un VLAN local :

25. Le VTEP reçoit la **trame Ethernet**
26. Il ajoute un **header VXLAN**
27. Il encapsule le tout dans **UDP + IP**
28. Le paquet est envoyé vers l'adresse IP du **VTEP distant**

La trame peut alors traverser un réseau **Layer 3 classique (Internet, WAN, backbone IP)**.

2️⃣ Décapsulation

Lorsque le VTEP distant reçoit le paquet :

25. Il retire l'en-tête **IP**
26. Il retire l'en-tête **UDP**
27. Il retire l'en-tête **VXLAN**
28. Il récupère la **trame Ethernet originale**

La trame est ensuite injectée dans le **VLAN local correspondant**.

Identification du réseau : le VNI

Chaque tunnel VXLAN est identifié par un **VNI (VXLAN Network Identifier)**.

- Taille : **24 bits**
- Nombre possible de réseaux : **≈ 16 millions**

C'est l'équivalent d'un **VLAN ID**, mais avec beaucoup plus de capacité.

Introduction du projet

Dans cette documentation, nous allons mettre en place un **tunnel VXLAN entre plusieurs machines Linux** afin de comprendre concrètement le fonctionnement de cette technologie de virtualisation réseau.

VXLAN (Virtual Extensible LAN) permet d'étendre un réseau de **couche 2 (Ethernet / VLAN)** à travers une infrastructure **de couche 3 (réseau IP)**. Cette technologie est largement utilisée dans les **datacenters, environnements cloud et infrastructures virtualisées**, car elle permet d'interconnecter des machines situées sur des réseaux physiques différents tout en conservant un même réseau logique.

L'objectif de ce projet est de **comprendre et implémenter VXLAN étape par étape** dans un environnement Linux.
Pour faciliter l'apprentissage et la compréhension, la mise en place sera réalisée de manière progressive, en augmentant progressivement la complexité du réseau.

Le projet sera organisé selon les étapes suivantes :

19. **Mise en place d'un VXLAN entre deux machines Linux situées sur le même réseau IP** afin de comprendre les bases du fonctionnement et de l'encapsulation.
20. **Mise en place d'un VXLAN entre deux machines situées sur des sous-réseaux différents**, séparés par un routeur, afin d'illustrer l'extension d'un réseau de couche 2 à travers une infrastructure de couche 3.
21. **Mise en place d'un VXLAN entre deux machines distantes à travers Internet**, afin de reproduire un scénario proche d'une architecture inter-datacenter.

À travers ces différentes étapes, nous allons analyser :

- le rôle des **VTEP (VXLAN Tunnel End Point)**
- le fonctionnement de l'**encapsulation VXLAN dans UDP/IP**
- la création d'interfaces VXLAN sous Linux
- et l'extension d'un réseau Ethernet au-dessus d'un réseau IP.

Cette approche progressive permettra de comprendre **à la fois les concepts théoriques et la mise en œuvre pratique de VXLAN** dans un environnement réel.

Les deux machines

VTEP1 : 192.168.0.33

VTEP2 : 192.168.0.34

Commande sur VTEP 1 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Commande sur VTEP 2 :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Sur VTEP1 :

```
root@grogu:~# ip linl set vxlan0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip linl set br0 up
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set br0 up
root@grogu:~# ip linl set vxlan0 master br0
Object "linl" is unknown, try "ip help".
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.1/24 dev br0
root@grogu:~#
```

1️⃣ Configuration VTEP1

créer l'interface VXLAN

ip link add vxlan0 type vxlan id 5000 dev eth0 remote 192.168.1.20 dstport 4789

explication :

option	**rôle**
id 5000	VNI
dev eth0	interface réseau physique
remote	IP du VTEP distant
dstport 4789	port VXLAN standard

---

activer l'interface

ip link set vxlan0 up

---

créer un bridge

ip link add br0 type bridge

---

activer le bridge

ip link set br0 up

---

connecter VXLAN au bridge

ip link set vxlan0 master br0

---

mettre l'IP overlay

ip addr add 10.10.10.1/24 dev br0

VTEP2 :

```
root@grogu:~# ip link set vxlan0 up
root@grogu:~# ip link add br0 type bridge
root@grogu:~# ip link set br0 up
root@grogu:~# ip link set vxlan master br0
Cannot find device "vxlan"
root@grogu:~# ip link set vxlan0 master br0
root@grogu:~# ip addr add 10.10.10.2/24 dev br0
root@grogu:~#
```

Test du ping :

Résolution d’un problème de connectivité VXLAN

Lors de la mise en place du tunnel VXLAN entre les deux machines Linux jouant le rôle de VTEP, le tunnel semblait fonctionner correctement au niveau du transport. En effet, l'analyse du trafic avec tcpdump montrait bien des paquets VXLAN encapsulés en UDP (port 4789) circulant entre les deux hôtes. Cela confirmait que le réseau underlay (réseau IP transportant VXLAN) fonctionnait correctement.

Cependant, malgré l’existence du tunnel, la connectivité au niveau du réseau overlay ne fonctionnait pas. Les tests de ping entre les deux adresses IP du réseau VXLAN (10.10.10.1 et 10.10.10.2) échouaient systématiquement.

L'analyse de la table des voisins (ip neigh show) montrait que les entrées ARP restaient dans l’état FAILED, ce qui indiquait que les requêtes ARP n’étaient pas correctement résolues. Autrement dit, les machines ne parvenaient pas à apprendre l’adresse MAC associée à l’adresse IP distante.

Après investigation, la cause du problème a été identifiée : les deux machines virtuelles avaient les mêmes adresses MAC sur certaines interfaces réseau. Cette situation est courante lorsque des machines virtuelles sont créées à partir d’un clone ou d’une image système identique. Dans ce cas, certaines interfaces réseau héritent de la même adresse MAC.

Dans un environnement de couche 2, comme celui créé par VXLAN, les adresses MAC jouent un rôle essentiel dans le fonctionnement du bridge. Le bridge agit comme un switch Ethernet, qui apprend dynamiquement quelles adresses MAC sont accessibles derrière chacun de ses ports. Si deux équipements présentent la même adresse MAC, l’apprentissage devient incohérent et le trafic peut être mal dirigé ou bloqué.

Pour résoudre ce problème, il a été décidé de supprimer les interfaces VXLAN et le bridge existants afin de repartir sur une configuration propre. Les interfaces ont ensuite été recréées en attribuant explicitement des adresses MAC différentes aux bridges et aux interfaces VXLAN de chaque VTEP.

Cette approche garantit que chaque interface possède une identité unique dans le réseau de couche 2, permettant ainsi au mécanisme d'apprentissage des adresses MAC du bridge de fonctionner correctement.

Une fois les interfaces recréées avec des adresses MAC distinctes et le tunnel VXLAN correctement configuré, les requêtes ARP ont pu circuler normalement à travers le tunnel. Les adresses MAC des hôtes ont alors été apprises par les bridges, ce qui a permis d'établir la connectivité IP entre les deux machines dans le réseau overlay.

Les tests de ping entre les adresses 10.10.10.1 et 10.10.10.2 ont alors réussi, confirmant le bon fonctionnement du tunnel VXLAN.

Voici une **section de documentation claire avec toutes les commandes**, séparées pour **VTEP1** et **VTEP2**, que tu peux copier dans ton rapport.

Configuration des VTEP

Dans ce laboratoire, deux machines Linux jouent le rôle de **VTEP (VXLAN Tunnel End Point)**.
Chaque VTEP possède :

- une interface réseau dans le **réseau underlay**
- une interface **VXLAN**
- un **bridge** permettant de relier le VXLAN au réseau overlay

Les adresses utilisées sont les suivantes :

Machine	IP Underlay	IP Overlay
VTEP1	192.168.0.33	10.10.10.1
VTEP2	192.168.0.34	10.10.10.2

Le **VXLAN Network Identifier (VNI)** utilisé est :

5000

Le port UDP standard de VXLAN est :

4789

Configuration du VTEP1

Suppression d'une éventuelle configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:01

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:01

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.1/24 dev br0

Ajout d'une entrée FDB pour permettre la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.34

Nettoyage du cache ARP :

ip neigh flush dev br0

Configuration du VTEP2

Suppression d'une configuration précédente :

ip link del vxlan0 2>/dev/null

ip link del br0 2>/dev/null

Création du bridge :

ip link add br0 type bridge

ip link set dev br0 address 02:00:00:00:10:02

ip link set br0 up

Création de l'interface VXLAN :

ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.33 dstport 4789

Définition d'une adresse MAC unique pour l'interface VXLAN :

ip link set dev vxlan0 address 02:00:00:00:20:02

Connexion du VXLAN au bridge :

ip link set vxlan0 master br0

Activation de l'interface VXLAN :

ip link set vxlan0 up

Configuration de l'adresse IP dans le réseau overlay :

ip addr add 10.10.10.2/24 dev br0

Ajout de l'entrée FDB pour la propagation des broadcasts :

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 192.168.0.33

Nettoyage du cache ARP :

ip neigh flush dev br0

Vérification du fonctionnement

Test de connectivité entre les deux VTEP :

ping 10.10.10.2

ou dans l'autre sens :

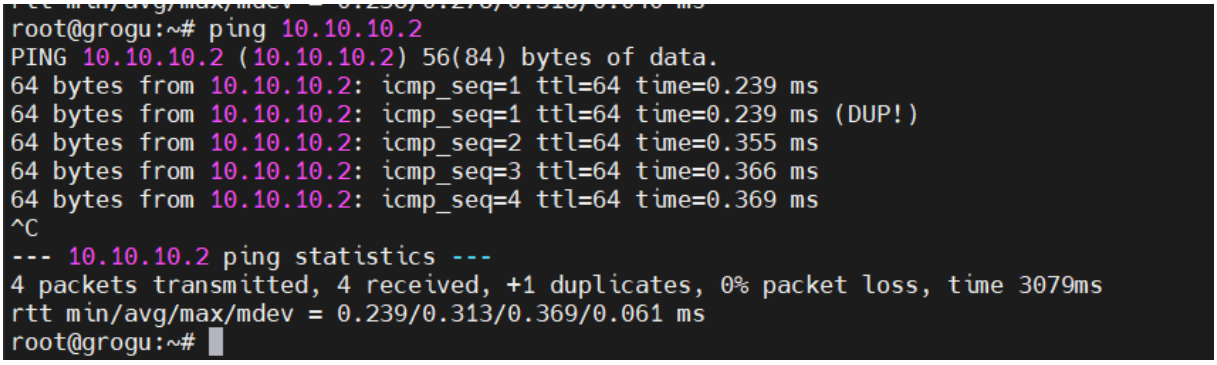
ping 10.10.10.1

Vérification du trafic VXLAN :

tcpdump -ni ens18 port 4789

Cette commande permet d'observer les paquets **VXLAN encapsulés en UDP** circulant entre les deux VTEP.

VTEP1 -> VTEP2

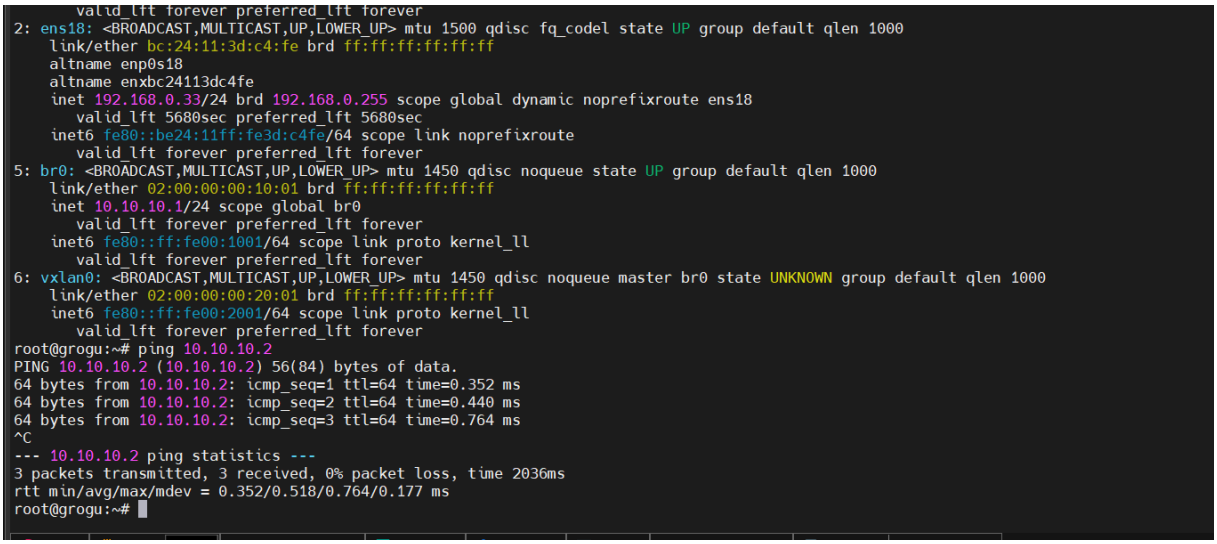


Ajout de 2 VM clientes et modification structure

Jusqua maintenant les deux VTP étaient dans le même vlan à la maison sois le 1 maintenant VTEP1 est dans le 1 et garde son ip 192.168.0.33

Le VTEP passe dans le VLAN 100 et aura comme IP : 10.0.0.147

Nous allons voir si le VXLAN fonctionne toujours



Le ping passe parfaitement

CLI-01 : 192.168.0.69 / derrière VTEP1 / 10.10.10.10

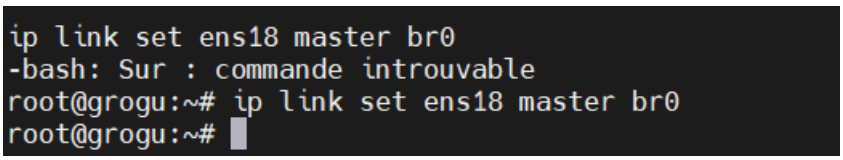
CLI-02 : 10.0.0.139/ derrière VTEP2 / 10.10.10.20

Sur les deux VTEP il faut faire un bridge entre le vxlan et l'interface brancher au switch

ERRER C'EST AVEC L INTERFACE COTE CLIENT QU IL FAUT FAIRE BRIDGE SOIS ENS19

Sur les deux VTEP :

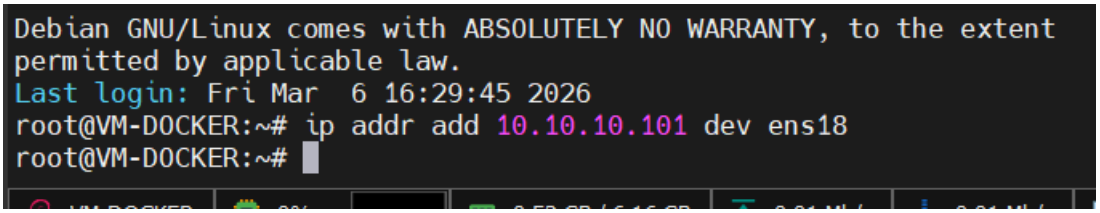
ip link set ens18 master br0



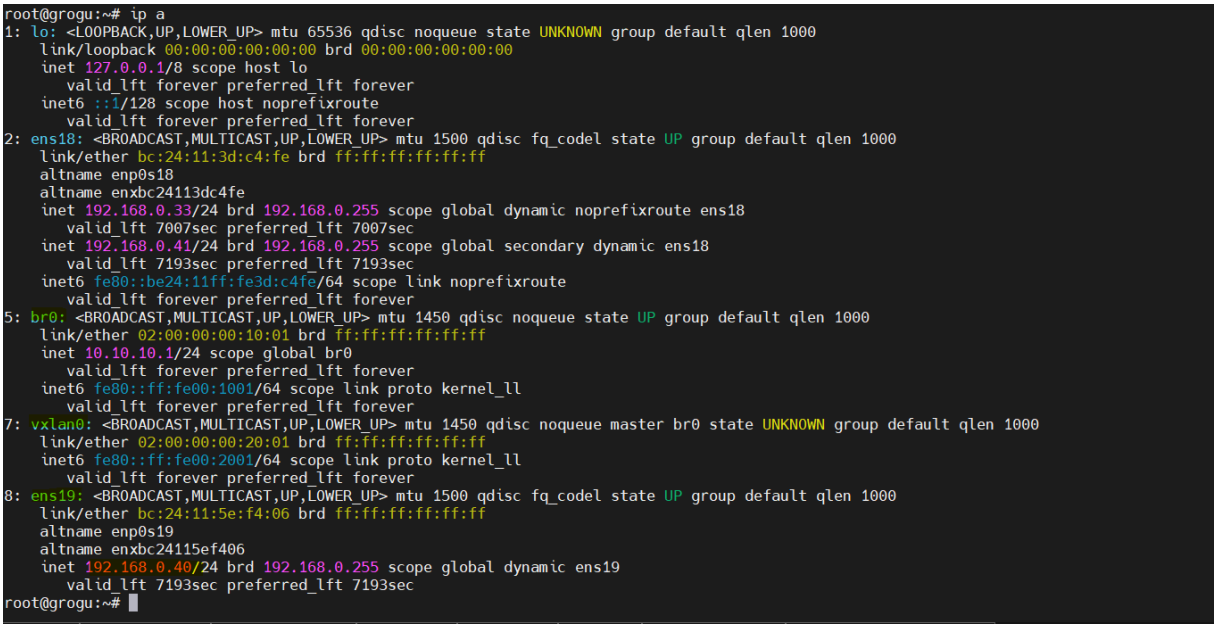
VM derrière VTEP 02 : 10.10.10.100

VM derrière VTEP 01 : 10.10.10.101

Voila comment on ajoute l'adresse



Pour que les VTEP puissent faire transiter des clients à travers notre VXLAN quand on a un serveur VTEP sous linux il faut absolument avoir une interface bridge qui combinera une seconde interface réseau dans le réseau des clients qui aura une IP dans e reseau en l'occurrence chez nous ens19 et dans le bridge il faudra avoir l'interface vxlan0 aussi les deux seront en mode bridge et pourront communiquer entre eux, vxlan0 captera les broascast de ens19



Comme on le voit ici j'ai rajouté ens19 et j avais déjà vxlan0

Sur le VTEP01 executer cette commande :

ip link set ens19 master br0

ip link set ens19 up

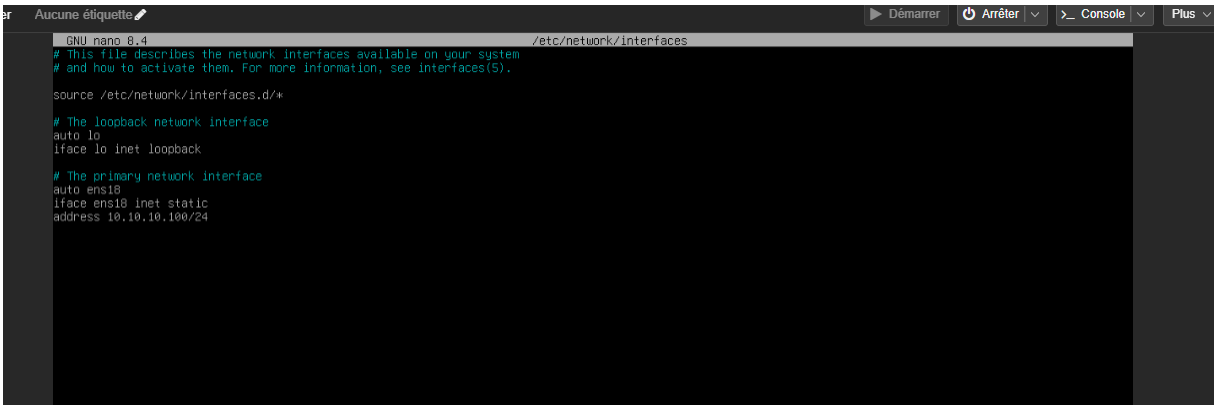
bridge link

```
root@grogu:~# ip link set ens19 master br0
ip link set ens19 up
root@grogu:~# bridge link
7: vxlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 master br0 state forwardin
g priority 32 cost 100
8: ens19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br0 state forwarding
 priority 32 cost 100
root@grogu:~#
```

Log entre les deux VTEP

```
root@grogu:~# tcpdump -ni ens18 port 4789
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:56:26.779234 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.779243 IP 192.168.0.33.53932 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::bad8:2dff:fe47:17a4.1900 > ff02::c.1900: UDP, length 618
20:56:26.962286 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:26.962295 IP 192.168.0.33.59591 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP6 fe80::ff:fe00:2001.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
20:56:28.177923 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.177931 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:28.835896 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:28.835903 IP 192.168.0.33.49759 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
10:e9:92:90:54:40 > ff:ff:ff:ff:ff:ff, ethertype Unknown (0x887b), length 60:
        0x0000:  0102 0400 0487 1000 0000 0000 0000 0000  ................
        0x0010:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0020:  0000 0000 0000 0000 0000 0000            ............
20:56:29.179332 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:29.179340 IP 192.168.0.33.43497 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
ARP, Request who-has 192.168.1.12 tell 192.168.1.254, length 28
20:56:30.152855 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152865 IP 192.168.0.33.32873 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.1.254.520 > 224.0.0.9.520: RIPv2, Response, length: 264
20:56:30.152871 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
IP 192.168.0.254.520 > 224.0.0.9.520: RIPv2, Response, length: 284
20:56:30.152880 IP 192.168.0.33.60687 > 10.0.0.149.4789: VXLAN, flags [I] (0x08), vni 5000
```

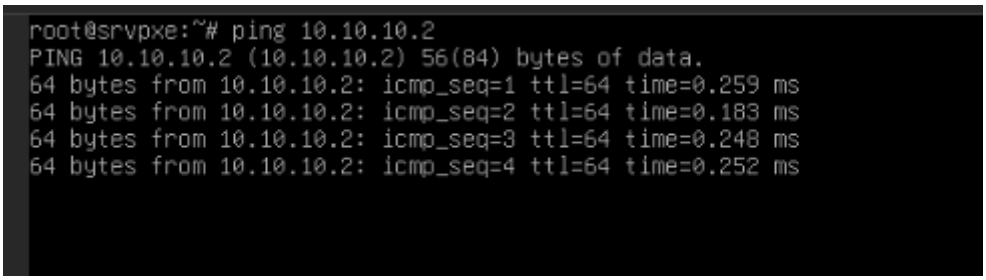
VM cliente dans VLAN 100 derriere VTEP 2



Je lui attribue une ip dans le overlay « 10.10.10.100 »

Je tente un ping vers le VTEP 2

Il passe avec succès



Vers VTEP1 distant

Il passe aussi avec succès

```
root@srvpxe:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.474 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=0.633 ms
64 bytes from 10.10.10.1: icmp_seq=6 ttl=64 time=0.613 ms
64 bytes from 10.10.10.1: icmp_seq=7 ttl=64 time=0.501 ms
64 bytes from 10.10.10.1: icmp_seq=8 ttl=64 time=0.565 ms
64 bytes from 10.10.10.1: icmp_seq=9 ttl=64 time=0.514 ms
64 bytes from 10.10.10.1: icmp_seq=10 ttl=64 time=0.605 ms
64 bytes from 10.10.10.1: icmp_seq=11 ttl=64 time=0.987 ms
```

Ping depuis VM 01 derriere VTEP 01

Adresse du client 01 : 10.10.10.200

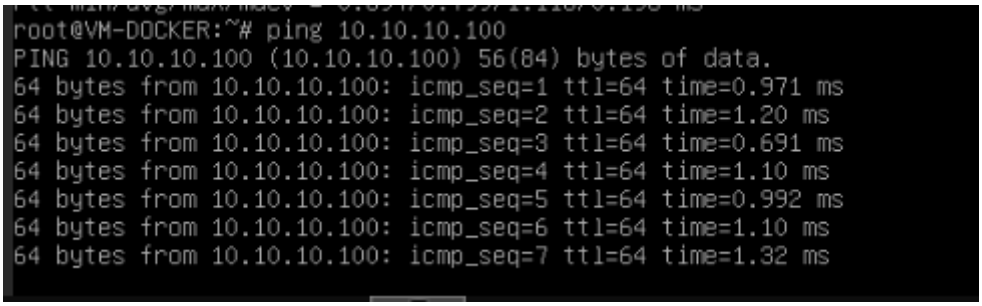
Ping vers VTEP 01

```
root@VM-DOCKER:~# ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=0.270 ms

64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=0.441 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=0.335 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=0.385 ms
^C
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 67ms
rtt min/avg/max/mdev = 0.270/0.357/0.441/0.067 ms
root@VM-DOCKER:~#
```

Ping vers VTEP 02 :

```
root@VM-DOCKER:~# ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.637 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=1.02 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=1.01 ms
```

Ping vers client 02 derriere VTEP 02



Parfait tout passe

Voici **une section prête à coller dans ton rapport** pour analyser les tables **FDB et ARP** de ton lab VXLAN.

Analyse des tables MAC et ARP dans le tunnel VXLAN

Afin de vérifier le bon fonctionnement du tunnel VXLAN et du bridge Linux, il est possible d'analyser les tables de commutation et de voisinage du système. Ces tables permettent de comprendre comment les adresses MAC et IP des machines sont apprises et utilisées pour acheminer le trafic dans le réseau overlay.

Deux commandes principales permettent cette analyse :

```
bridge fdb show
```

et

```
ip neigh
```

Analyse de la table FDB (Forwarding DataBase)

La commande suivante permet d'afficher la table de commutation du bridge :

```
bridge fdb show
```

Cette table est équivalente à la table MAC d'un switch Ethernet.
Elle indique pour chaque adresse MAC sur quel port du bridge le trafic doit être envoyé.

La structure de la table est la suivante :

```
adresse MAC → interface du bridge
```

Dans notre cas, le bridge **br0** relie deux interfaces :

- **ens19** : interface connectée au réseau des machines clientes
- **vxlan0** : interface du tunnel VXLAN

Apprentissage des adresses MAC locales

Exemple d'entrées observées :

```
96:12:a6:1a:99:3c dev ens19 master br0
bc:24:11:12:26:b1 dev ens19 master br0
5e:89:38:5d:91:b0 dev ens19 master br0
```

Ces adresses MAC ont été apprises sur l'interface **ens19**.
Cela signifie que ces machines sont connectées localement derrière le VTEP.

Le bridge apprend automatiquement ces adresses lorsqu'il reçoit des trames Ethernet provenant de ces équipements.

Schéma simplifié :

```
VM cliente → ens19 → br0
```

Apprentissage des adresses MAC distantes via VXLAN

D'autres entrées apparaissent sur l'interface **vxlan0** :

```
bc:24:11:3e:82:42 dev vxlan0 master br0
bc:24:11:c7:fb:75 dev vxlan0 master br0
bc:24:11:d3:4b:86 dev vxlan0 master br0
```

Ces adresses MAC correspondent aux machines situées derrière le **VTEP distant**.

Le bridge apprend ces adresses lorsque des trames arrivent via le tunnel VXLAN.

Schéma :

```
VM distante → VTEP distant → VXLAN → vxlan0 → br0
```

Cela montre que le bridge Linux se comporte comme un **switch Ethernet logiciel**, capable d'apprendre dynamiquement les adresses MAC situées de part et d'autre du tunnel VXLAN.

Entrée spéciale de flooding VXLAN

Une entrée particulière peut également être observée :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

Cette entrée correspond à la règle de **flooding** utilisée par VXLAN.

Elle permet d'envoyer les trames de type :

- broadcast
- multicast
- unknown unicast

vers le VTEP distant.

Cela est indispensable pour permettre le fonctionnement de protocoles comme :

- ARP
- DHCP
- découverte des voisins

Sans cette entrée, certaines trames nécessaires au fonctionnement du réseau overlay ne pourraient pas être transmises dans le tunnel VXLAN.

Analyse de la table ARP

La commande suivante permet d'afficher la table ARP du système :

```
ip neigh
```

Cette table associe les **adresses IP aux adresses MAC correspondantes**.

Exemple observé :

```
10.10.10.100 dev br0 lladdr bc:24:11:c7:fb:75 REACHABLE
```

Cela signifie :

Champ	Signification
IP	10.10.10.100
MAC	bc:24:11:c7:fb:75
Interface	br0

Cette machine appartient au **réseau overlay VXLAN**.

Exemple de VTEP distant

Une autre entrée montre la présence du VTEP distant dans le réseau overlay :

```
10.10.10.2 dev br0 lladdr 02:00:00:00:10:02
```

Cette entrée indique que l'adresse IP overlay **10.10.10.2** correspond à l'adresse MAC du bridge du VTEP distant.

Les requêtes ARP sont envoyées sous forme de broadcast dans le tunnel VXLAN, permettant aux VTEP d'apprendre les correspondances IP/MAC.

Exemple de machine cliente distante

On observe également :

```
10.10.10.200 dev br0 lladdr bc:24:11:12:26:b1
```

Cette machine est située derrière le VTEP distant mais apparaît dans la table ARP locale grâce au tunnel VXLAN.

Le processus est le suivant :

31. une requête ARP est envoyée dans le réseau overlay
32. elle est encapsulée dans VXLAN
33. elle est transmise au VTEP distant
34. la machine distante répond
35. l'adresse MAC est enregistrée dans la table ARP

Conclusion

L'analyse des tables FDB et ARP confirme le bon fonctionnement du tunnel VXLAN.

Les résultats montrent que :

- le bridge Linux apprend les adresses MAC des machines locales sur l'interface **ens19**
- les adresses MAC des machines distantes sont apprises sur **vxlan0**
- les requêtes ARP sont correctement transmises dans le tunnel VXLAN
- les machines situées sur des réseaux physiques différents peuvent communiquer comme si elles appartenaient au même réseau Ethernet.

Cette observation confirme que VXLAN permet bien d'étendre un réseau de couche 2 au-dessus d'une infrastructure de couche 3.

Si tu veux, je peux aussi te faire **la dernière section parfaite pour ton rapport** :

- **Analyse complète d'un paquet VXLAN avec tcpdump**
- **schéma overlay / underlay propre**
- **explication du VNI**

Ça ferait une **fin de doc très propre niveau ingénieur réseau**.

Introduction à VXLAN EVPN

Dans les sections précédentes, nous avons mis en place un tunnel VXLAN entre plusieurs machines Linux afin d'étendre un réseau de couche 2 au-dessus d'une infrastructure IP de couche 3. Cette approche permet de connecter des machines situées sur des réseaux physiques différents tout en leur donnant l'illusion d'appartenir au même réseau Ethernet.

Dans cette configuration initiale, le fonctionnement du VXLAN repose sur un mécanisme appelé **Flood & Learn**. Lorsqu'une machine souhaite communiquer avec une autre dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée sous forme de broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet à tous les VTEP participants au tunnel. Une fois la réponse reçue, le bridge apprend dynamiquement l'association entre l'adresse MAC et l'interface correspondante.

Bien que ce mécanisme soit simple et efficace dans des environnements de petite taille, il présente plusieurs limites lorsqu'il est utilisé dans des infrastructures de grande envergure. Dans un datacenter moderne, il peut exister plusieurs centaines de VTEP et plusieurs milliers de machines virtuelles. Dans ce contexte, le mécanisme de flooding génère un volume important de trafic broadcast, ce qui peut provoquer une surcharge du réseau et limiter la scalabilité de l'architecture.

Pour résoudre ce problème, les infrastructures modernes utilisent un **plan de contrôle EVPN (Ethernet VPN)**. EVPN repose sur le protocole BGP pour distribuer les informations de couche 2 entre les VTEP. Au lieu d'apprendre les adresses MAC par diffusion, les VTEP annoncent directement dans BGP les informations relatives aux machines présentes derrière eux, notamment l'adresse MAC, l'adresse IP associée et le VNI auquel elles appartiennent.

Ainsi, lorsqu'une machine apparaît dans le réseau, son VTEP annonce ces informations aux autres VTEP via BGP EVPN. Les VTEP distants savent alors immédiatement vers quel VTEP envoyer le trafic destiné à cette machine, sans avoir besoin d'utiliser de mécanisme de flooding.

L'utilisation d'EVPN apporte plusieurs avantages majeurs :

- réduction drastique du trafic broadcast dans le réseau overlay
- apprentissage des adresses MAC via un plan de contrôle centralisé

- meilleure scalabilité pour les environnements comportant un grand nombre de machines virtuelles
- possibilité d'implémenter des fonctionnalités avancées comme la suppression ARP ou le routage inter-VXLAN

Cette approche est aujourd'hui largement utilisée dans les architectures de datacenter modernes, notamment dans les fabric **leaf-spine**, où les VTEP échangent leurs informations via BGP EVPN afin de construire une infrastructure réseau hautement scalable et performante.

Dans la section suivante, nous allons étendre notre laboratoire VXLAN en introduisant **BGP EVPN** à l'aide de FRRouting. Cette étape permettra de remplacer le mécanisme de Flood & Learn par un plan de contrôle basé sur BGP, similaire à celui utilisé dans les infrastructures réseau des datacenters modernes.

Installer FRRouting

Sur **les deux VTEP**.

```
apt update
apt install frr -y
```

Activer les démons nécessaires

Éditer :

```
/etc/frr/daemons
```

Changer :

```
bgpd=yes
zebra=yes
```

Donc :

```
zebra=yes
bgpd=yes
```

Sur VTEP1

```
conf t
```

Créer BGP :

```
router bgp 65001
```

Router ID :

```
bgp router-id 192.168.0.33
```

Déclarer le voisin :

```
neighbor 192.168.0.34 remote-as 65001
neighbor 192.168.0.34 update-source ens18
```

Activer EVPN :

```
address-family l2vpn evpn
 neighbor 192.168.0.34 activate
 advertise-all-vni
exit-address-family
```

Sur VTEP 02

grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# bgp router-id 192.168.0.33

grogu(config-router)# no neighbor 192.168.0.33

grogu(config-router)# neighbor 10.0.0.139

% Command incomplete: neighbor 10.0.0.139

grogu(config-router)# bgp router-id 10.0.0.139

grogu(config-router)# neighbor 10.0.0.139 remote-as 65001

% Can not configure the local system as neighbor

grogu(config-router)# neighbor 192.168.0.33 remote-as 65001

grogu(config-router)# neighbor 192.168.0.33 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 192.168.0.33 activate

grogu(config-router-af)# advertise-all-vni

grogu(config-router-af)# exit-address-family

grogu(config-router)# show bgp suùùary

% Unknown command: show bgp suùùary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# show bgp summary

% Unknown command: show bgp summary

grogu(config-router)# exit

grogu(config)# exit

Vérifier que BGP fonctionne et est établie

On voit que sur vtep 01 le bgp fonctionne

```
grogu# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8        0    0    0 00:02:08            0        0 FRRouting/10.3

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 192.168.0.33, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 1, using 23 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.139      4      65001         7         8       20    0    0 00:02:08            6       16 FRRouting/10.3

Total number of neighbors 1
grogu#
```

Il est établie depuis 2 minutes

La partie importante : EVPN

Dans la deuxième section :

`L2VPN EVPN Summary`

On voit :

`State/PfxRcd   16`

Ça veut dire :

```
16 routes EVPN reçues
```

Donc ton VTEP reçoit des informations comme :

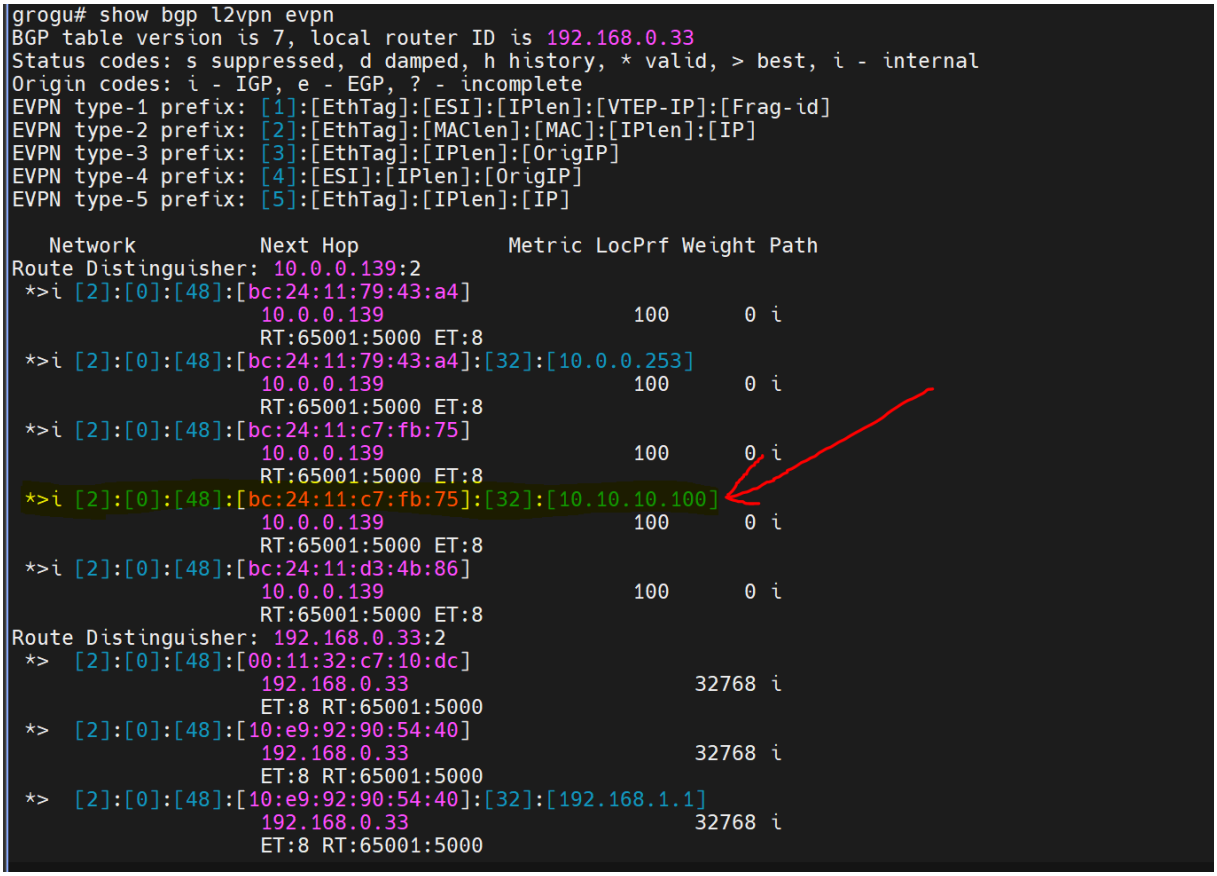
```
MAC
IP
VNI
VTEP source
```

via BGP.

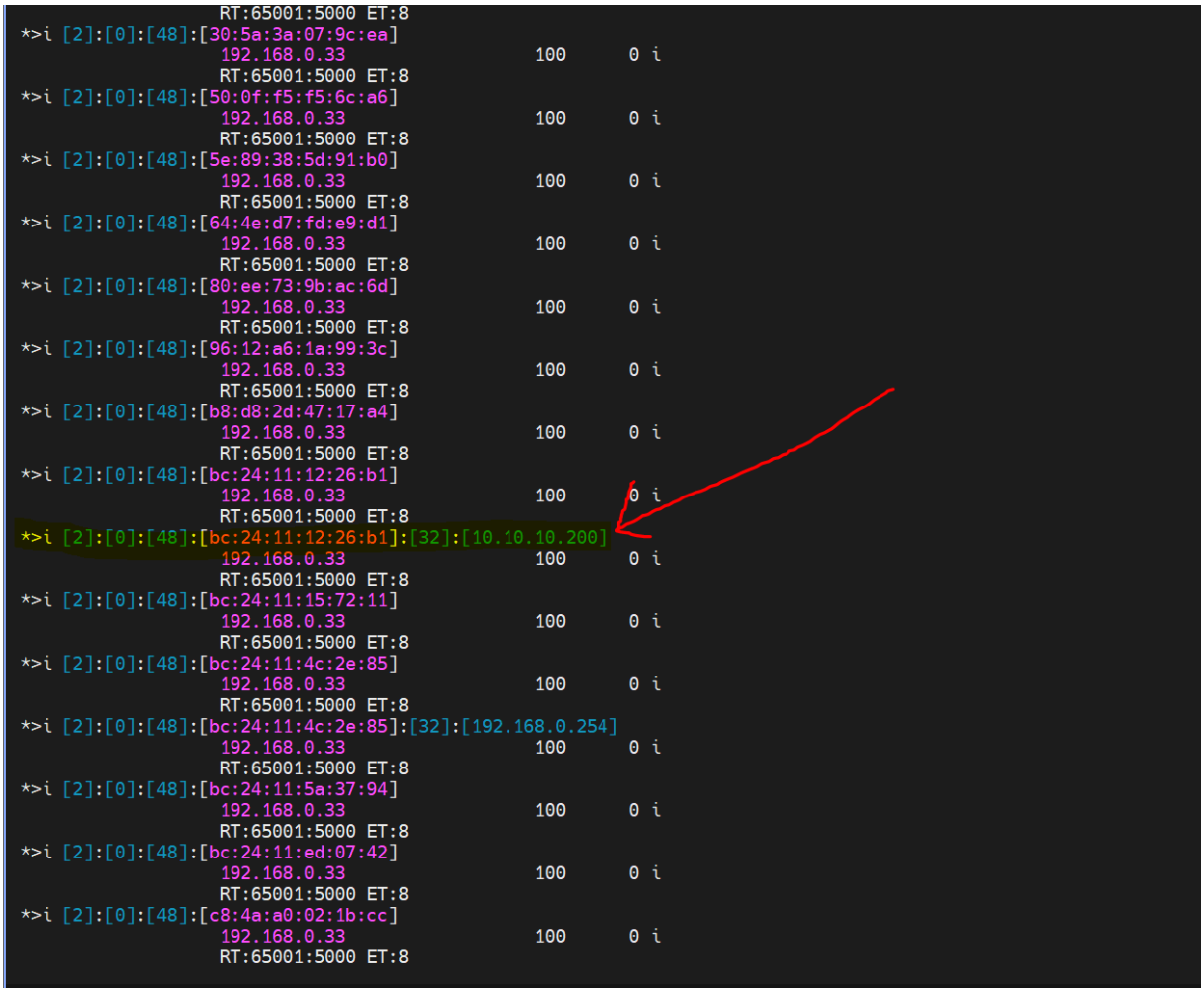
Afficher les routes EVPN

show bgp l2vpn evpn

On voit ici la mac de la VM derriere le VTEP 02 est bien envoyé par VTEP2 et appris par VTEP1



Coté VTEP02



On voit bien la VM derriere VTEP 1

ARP SUPPRESSION

On va faire en sorte que les VTEP n'envoient plus de broadcast pour connaitre une IP mais qu'ils se la communqiuent directement

Activer ARP suppression

Dans FRRouting :

ouvre :

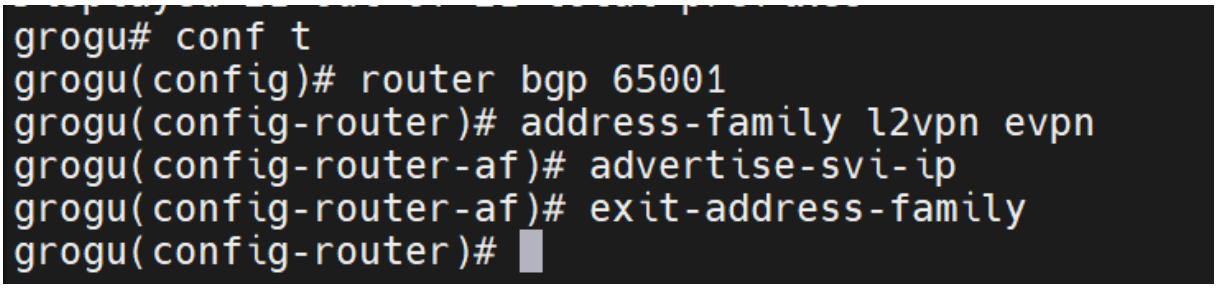
```
vtysh
```

Sur chaque VTEP

```
conf t
```

puis :

```
router bgp 65001
address-family l2vpn evpn
 advertise-svi-ip
exit-address-family
```



TEST

Sur les VTEP — vider les MAC du bridge

Sur chaque VTEP :

```
bridge fdb flush br0
```

ou plus ciblé :

```
bridge fdb flush dev vxlan0
```

Ça vide :

```
MAC → VTEP
```

root@grogu:~# bridge fdb flush dev br0

root@grogu:~# bridge fdb flush dev vxlan0

ip neigh flush all

Il faut des deux cotés des VTEP impérativement ajouter l'entréé statique dans la table MAC

bridge fdb append 00:00:00:00:00:00 dev vxlan0 dst 10.0.0.139

Sur **chaque VTEP**, active ça :

```
bridge link set dev vxlan0 neigh_suppress on
```

Puis vérifie :

```
bridge -d link show dev vxlan0
```

Tu dois voir `neigh_suppress on`.

bridge link set dev vxlan0 neigh_suppress on

bridge link set dev vxlan0 learning off

bridge -d link show dev vxlan0

Mise en place de VXLAN EVPN avec FRRouting

Introduction à EVPN

Dans les étapes précédentes, le tunnel VXLAN mis en place entre les VTEP fonctionnait selon le mécanisme classique de **Flood and Learn**.
Dans ce mode, lorsqu'une machine souhaite joindre une autre machine dont elle ne connaît pas encore l'adresse MAC, une requête ARP est envoyée en broadcast dans le réseau overlay. Le VTEP encapsule alors cette requête dans VXLAN et la transmet au VTEP distant. Une fois la réponse reçue, le bridge Linux apprend dynamiquement l'association entre l'adresse MAC et l'interface concernée.

Cette méthode fonctionne correctement dans un petit laboratoire, mais elle présente plusieurs limites dans une architecture de plus grande taille :

- elle génère du trafic broadcast
- elle repose sur l'apprentissage dynamique par le bridge
- elle n'est pas optimale pour des environnements comportant de nombreux VTEP

Pour améliorer cela, nous avons mis en place un **plan de contrôle EVPN (Ethernet VPN)** reposant sur **BGP** grâce à **FRRouting (FRR)**.

Le principe d'EVPN est de ne plus laisser uniquement le bridge Linux apprendre les adresses MAC par inondation du trafic, mais de faire en sorte que les VTEP s'échangent directement les informations sur les machines présentes derrière eux via BGP.

Ainsi, chaque VTEP peut annoncer :

- les adresses MAC apprises localement
- les adresses IP associées
- le VNI concerné
- le VTEP derrière lequel se trouve la machine

Cette approche permet de remplacer progressivement le comportement Flood and Learn par un apprentissage via le plan de contrôle.

Installation de FRRouting

Sur les deux VTEP, nous avons installé FRRouting avec la commande suivante :

```
apt update
apt install frr -y
```

FRR est une suite logicielle permettant d'implémenter plusieurs protocoles de routage sur Linux, dont BGP.
Dans notre cas, il a été utilisé pour mettre en œuvre **BGP EVPN**.

Activation des démons nécessaires

Le fichier suivant a été modifié :

```
/etc/frr/daemons
```

Les options suivantes ont été activées :

```
zebra=yes
```

```
bgpd=yes
```

Explication :

- **zebra** permet à FRR d'interagir avec le noyau Linux et les interfaces réseau
- **bgpd** active le démon BGP, nécessaire pour EVPN

Après modification, le service FRR a été redémarré :

```
systemctl restart frr
```

Configuration BGP EVPN

La configuration a été réalisée via l'interface :

```
vtysh
```

Cette commande permet d'entrer dans l'environnement de configuration FRR.

Configuration du VTEP1

Sur le VTEP1, la configuration suivante a été appliquée :

```
conf t
router bgp 65001
 bgp router-id 192.168.0.33
 neighbor 10.0.0.145 remote-as 65001
 neighbor 10.0.0.145 update-source ens18
 address-family l2vpn evpn
  neighbor 10.0.0.145 activate
  advertise-all-vni
 exit-address-family
exit
write
```

Explication des commandes

`conf t`

Permet d'entrer en mode configuration.

`router bgp 65001`

Crée un processus BGP local dans l'AS **65001**.

bgp router-id 192.168.0.33

Définit l'identifiant du routeur BGP.
Il s'agit d'un identifiant logique permettant de distinguer le VTEP dans le plan de contrôle.

neighbor 10.0.0.145 remote-as 65001

Déclare le voisin BGP distant.
Le VTEP distant utilise lui aussi l'AS 65001, il s'agit donc d'une session **iBGP**.

neighbor 10.0.0.145 update-source ens18

Indique à FRR d'utiliser l'interface underlay `ens18` pour établir la session BGP.

address-family l2vpn evpn

Active la famille d'adresses EVPN.
C'est dans cette section que l'on configure les échanges de routes EVPN.

neighbor 10.0.0.145 activate

Active EVPN pour ce voisin.

advertise-all-vni

Indique à FRR d'annoncer tous les VNI détectés dans le dataplane Linux.

write

Sauvegarde la configuration.

Configuration du VTEP2

Sur le second VTEP, la configuration a été adaptée comme suit :

```
conf t
router bgp 65001
 bgp router-id 10.0.0.145
 neighbor 192.168.0.33 remote-as 65001
 neighbor 192.168.0.33 update-source ens18
 address-family l2vpn evpn
  neighbor 192.168.0.33 activate
  advertise-all-vni
 exit-address-family
exit
write
```

Le principe est exactement le même, avec inversion des adresses IP des deux VTEP.

Vérification de la session BGP EVPN

Une fois la configuration appliquée, la session BGP a été vérifiée avec la commande :

```
show bgp l2vpn evpn summary
```

Cette commande permet de vérifier :

- que la session BGP est bien établie
- que les deux VTEP communiquent en EVPN
- le nombre de routes EVPN reçues

Lorsque la session est fonctionnelle, on observe un état de type :

```
Established
```

et un compteur de routes reçues non nul.

Cela signifie que les VTEP s'échangent bien des informations EVPN.

Consultation des routes EVPN

Les routes EVPN ont été affichées avec :

```
show bgp l2vpn evpn route
```

Cette commande montre les routes EVPN apprises via BGP.

Dans notre lab, on a pu observer des routes de type 2, qui contiennent :

- l'adresse MAC d'une machine
- parfois l'adresse IP associée
- le VTEP derrière lequel se trouve cette machine

Cela a permis de constater que la MAC de la machine située derrière le VTEP2 était bien annoncée au VTEP1, et inversement.

Autrement dit, les VTEP n'apprennent plus uniquement les adresses par le trafic, mais aussi par le control plane EVPN.

Adaptation du dataplane Linux pour EVPN

Pour que FRR reconnaisse correctement le VNI 5000 et puisse l'utiliser dans EVPN, l'interface VXLAN a été recréée en mode compatible EVPN.

Sur le VTEP concerné, la commande suivante a été utilisée :

```
ip link del vxlan0
ip link add vxlan0 type vxlan \
 id 5000 \
 local 10.0.0.145 \
 dstport 4789 \
 dev ens18 \
 nolearning
ip link set vxlan0 up
ip link set vxlan0 master br0
```

Explication

`ip link del vxlan0`

Supprime l'ancienne interface VXLAN.

`ip link add vxlan0 type vxlan ...`

Crée une nouvelle interface VXLAN.

`id 5000`

Définit le VNI utilisé.

`local 10.0.0.145`

Spécifie l'adresse IP underlay locale du VTEP.

`dstport 4789`

Utilise le port UDP standard de VXLAN.

`dev ens18`

Utilise l'interface physique underlay `ens18`.

`nolearning`

Désactive l'apprentissage MAC local dans le driver VXLAN.
Cela est important en EVPN, car ce n'est plus le VXLAN qui doit apprendre les adresses MAC, mais le control plane EVPN.

Après cette recréation, FRR a enfin reconnu correctement le VNI.

Vérification du VNI EVPN

La commande suivante a permis de vérifier que FRR voyait bien le VNI :

```
show evpn vni
```

Exemple de résultat :

```
5000       L2   vxlan0                17       4        1
default
```

Cela signifie :

- le **VNI 5000** est actif
- il est associé à l'interface **vxlan0**
- FRR connaît des adresses MAC et ARP dans ce VNI
- un VTEP distant est vu dans ce VNI

Vérification des adresses MAC EVPN

La commande suivante a ensuite été utilisée :

```
show evpn mac vni 5000
```

Cette commande affiche les adresses MAC connues dans le VNI 5000.

Elle permet de distinguer :

- les MAC **locales**, apprises sur `ens19`
- les MAC **distantes**, apprises via EVPN depuis le VTEP voisin

Exemple d'interprétation :

- une MAC marquée **local** correspond à une machine derrière le VTEP local
- une MAC marquée **remote** correspond à une machine annoncée par le VTEP distant

Cette commande a permis de confirmer que le plan de contrôle EVPN fonctionnait correctement.

Vérification des entrées ARP EVPN

La commande suivante a été utilisée :

```
show evpn arp-cache vni 5000
```

Elle affiche les correspondances IP / MAC connues dans le VNI EVPN.

Exemple observé :

```
10.10.10.100    local   active   bc:24:11:c7:fb:75
```

Cela signifie que le VTEP a bien appris :

- l'adresse IP `10.10.10.100`
- l'adresse MAC associée
- l'état actif de l'entrée

Cette étape est importante, car en EVPN une machine n'est pas réellement exploitable tant que le VTEP n'a pas appris son couple IP/MAC et ne l'a pas injecté dans le control plane.

Tests réalisés

Test entre les VTEP overlay

Le test suivant a été réalisé :

```
ping -I br0 -c 3 10.10.10.1
```

Résultat :

- la connectivité entre les deux VTEP overlay est fonctionnelle
- le dataplane VXLAN EVPN est bien opérationnel

Test vers une machine cliente

Un test a ensuite été réalisé vers la machine cliente `10.10.10.100`.

Au départ, le ping ne fonctionnait pas, car la machine n'avait pas encore été apprise dans le control plane EVPN.

Après génération de trafic depuis la machine elle-même vers le VTEP, son VTEP local a appris :

- sa MAC
- son IP
- et a injecté ces informations dans EVPN

La machine est alors apparue dans :

```
show evpn arp-cache vni 5000
```

Une fois cette étape effectuée, la connectivité a pu être validée.

Tentative de suppression ARP

L'objectif suivant était de vérifier si les VTEP pouvaient éviter de transmettre les requêtes ARP dans le tunnel grâce à EVPN.

Pour cela, les commandes suivantes ont été utilisées dans FRR :

```
conf t
router bgp 65001
 address-family l2vpn evpn
  advertise-svi-ip
 exit-address-family
exit
write
```

Explication

advertise-svi-ip

Cette commande permet d'annoncer les adresses IP des interfaces de type SVI / bridge dans EVPN.
Elle est utilisée pour enrichir les annonces EVPN en informations IP.

Ensuite, côté Linux, les options suivantes ont été activées :

```
bridge link set dev vxlan0 neigh_suppress on
bridge link set dev vxlan0 learning off
bridge -d link show dev vxlan0
```

Explication

neigh_suppress on

Active la suppression des requêtes de voisinage sur le port bridge concerné.
Dans notre cas, cela vise à empêcher que certaines requêtes ARP traversent inutilement VXLAN.

learning off

Désactive l'apprentissage MAC classique sur le port VXLAN du bridge.

bridge -d link show dev vxlan0

Permet de vérifier les paramètres avancés du port `vxlan0`.

Résultat du test ARP suppression

Pour tester ce comportement, les caches ARP ont été vidés :

```
ip neigh flush all
```

Puis le trafic ARP a été observé avec :

```
tcpdump -n -i vxlan0 arp
```

Les captures ont montré que des requêtes ARP continuaient encore à traverser le tunnel VXLAN, par exemple :

- `ARP who-has 10.10.10.200 tell 10.10.10.100`
- `ARP who-has 10.10.10.100 tell 10.10.10.200`

Cela montre que, dans notre environnement de lab, l'ARP suppression n'a pas été totalement effective.

La raison principale est que le segment client branché sur `ens19` est encore bruité et transporte d'autres protocoles ou ARP qui polluent le test.
Le control plane EVPN fonctionne bien, mais le dataplane Linux n'a pas totalement supprimé le passage de certaines requêtes ARP dans le tunnel.

Conclusion sur la partie EVPN

Cette étape a permis de faire évoluer le lab d'un VXLAN classique en **VXLAN EVPN avec FRRouting**.

Les résultats obtenus montrent que :

- la session **BGP EVPN** entre les deux VTEP fonctionne
- le **VNI 5000** est bien reconnu par FRR
- les adresses **MAC locales et distantes** sont visibles dans EVPN
- les entrées **ARP/IP** sont connues par le control plane

- la connectivité entre les VTEP overlay est opérationnelle
- les machines clientes peuvent être apprises et annoncées via EVPN
- la suppression ARP a été partiellement testée, mais reste limitée dans ce lab à cause du bruit du réseau client

Cette évolution permet de mieux comprendre la différence entre :

- un VXLAN fonctionnant en **Flood and Learn**
- un VXLAN fonctionnant avec un véritable **control plane EVPN**

Dans le second cas, les VTEP ne se contentent plus de relayer du trafic broadcast : ils s'échangent directement les informations de couche 2 via BGP, ce qui correspond beaucoup plus aux architectures utilisées dans les datacenters modernes.

Voici un **court morceau de documentation** que tu peux insérer dans ton rapport pour répondre à cette question.

Apprentissage des adresses MAC malgré l'option *nolearning*

Lors de la création de l'interface VXLAN utilisée dans notre laboratoire, l'option `nolearning` a été activée :

```
ip link add vxlan0 type vxlan id 5000 dev ens18 local 10.0.0.145 dstport
4789 nolearning
```

Cette option peut sembler paradoxale : si l'apprentissage est désactivé, comment les VTEP peuvent-ils apprendre les adresses MAC des machines situées derrière eux ?

La réponse vient du fait que **l'apprentissage MAC n'est pas effectué par l'interface VXLAN elle-même**, mais par le **bridge Linux**.

Dans notre architecture, le bridge `br0` relie deux interfaces :

- `ens19` : interface connectée au réseau des machines clientes
- `vxlan0` : interface du tunnel VXLAN

Le fonctionnement est alors le suivant :

1. Une machine virtuelle locale envoie une trame Ethernet.
2. Cette trame arrive sur l'interface `ens19`.
3. Le bridge Linux `br0` apprend automatiquement l'adresse MAC source sur ce port.
4. Cette adresse MAC est enregistrée dans la **table FDB (Forwarding Database)** du bridge.

La commande suivante permet d'observer cette table :

```
bridge fdb show
```

On peut alors voir des entrées du type :

```
bc:24:11:c7:fb:75 dev ens19 master br0
```

Cela signifie que l'adresse MAC de la machine est accessible via l'interface `ens19`.

Le démon **FRRouting (FRR)** surveille ensuite la table FDB du bridge via le mécanisme **Netlink** du noyau Linux.
Lorsqu'une nouvelle adresse MAC est apprise localement, FRR récupère cette information et l'annonce aux autres VTEP via **BGP EVPN** sous forme de **routes EVPN de type 2 (MAC/IP Advertisement)**.

Ainsi, même si l'interface VXLAN est configurée avec `nolearning`, les adresses MAC des machines locales sont toujours apprises par le bridge Linux puis distribuées dans le plan de contrôle EVPN.

En résumé :

- `nolearning` désactive l'apprentissage MAC **dans l'interface VXLAN**
- l'apprentissage MAC est toujours réalisé par **le bridge Linux**
- FRR récupère ces informations et les **annonce via BGP EVPN**

Cette séparation permet de basculer d'un fonctionnement **Flood & Learn** vers un apprentissage contrôlé par le **plan de contrôle EVPN**, ce qui améliore la scalabilité et réduit le trafic broadcast dans le réseau overlay.

Très bonne question — c'est **le cœur du fonctionnement EVPN**, et c'est normal que ce soit confus parce que le mécanisme est indirect.
Je t'explique **précisément à quel moment les VTEP se "découvrent"** et **quelle commande déclenche cela**.

1️⃣ Dans VXLAN classique : découverte par configuration

Au début de ton lab tu avais :

```
ip link add vxlan0 type vxlan id 5000 dev ens18 remote 192.168.0.34 dstport 4789
```

Donc :

```
VTEP1 connaît VTEP2 parce qu'on l'a configuré
```

Le kernel sait directement :

```
envoyer VXLAN → 192.168.0.34
```

Limite :

```
1 remote par interface
```

Ce modèle **ne scale pas**.

2️⃣ En EVPN : découverte via BGP

Quand on passe en EVPN, on **supprime le remote** :

```
ip link add vxlan0 type vxlan id 5000 local 10.0.0.145 dev ens18 dstport
4789 nolearning
```

Donc maintenant :

```
vxlan0 ne sait plus vers qui envoyer
```

C'est **BGP EVPN qui va apprendre les autres VTEP**.

3️⃣ La commande qui déclenche ça

La commande clé est :

```
advertise-all-vni
```

dans :

```
router bgp 65001
 address-family l2vpn evpn
  advertise-all-vni
```

Cette commande dit à FRR :

annonce dans EVPN que **ce VTEP possède le VNI 5000**

4️⃣ Ce que FRR annonce réellement

Quand `advertise-all-vni` est actif, FRR regarde les interfaces VXLAN du système.

Il voit par exemple :

```
vxlan0
VNI 5000
local VTEP = 10.0.0.145
```

FRR envoie alors une **route EVPN type 3**.

Cette route s'appelle :

```
IMET route
(Inclusive Multicast Ethernet Tag)
```

Elle contient :

```
VNI
IP du VTEP
```

Exemple :

```
VNI: 5000
VTEP: 10.0.0.145
```

5️⃣ Ce que reçoit l'autre VTEP

Le VTEP distant reçoit cette route EVPN via BGP.

Il comprend :

```
le VNI 5000 existe aussi derrière ce VTEP
```

Donc il ajoute dynamiquement une entrée VXLAN.

Dans le kernel Linux ça devient :

```
vxlan0 → dst 10.0.0.145
```

Sans qu'on ait configuré `remote`.

6️⃣ Où ça apparaît dans Linux

Regarde :

```
bridge fdb show
```

Tu verras une entrée du type :

```
00:00:00:00:00:00 dev vxlan0 dst 10.0.0.145
```

C’est **la route IMET EVPN installée dans le kernel**.

Elle permet :

```
broadcast
unknown unicast
ARP
```

vers ce VTEP.

---

7 Ensuite les MAC arrivent

Après la découverte des VTEP :

1 chaque VTEP annonce ses hôtes via **EVPN route type 2**

```
MAC
IP
VNI
VTEP
```

2 les autres VTEP installent :

```
MAC → VTEP
```

Donc quand une VM envoie un paquet :

```
dest MAC connue → VXLAN vers le bon VTEP
```

---

8 Résumé du processus complet

étape 1

BGP EVPN s’établit :

```
neighbor X remote-as 65001
```

étape 2

FRR annonce les VNI :

```
advertise-all-vni
```

→ route EVPN type 3

```
VNI → VTEP
```

étape 3

Les VTEP se découvrent :

```
VTEP1 apprend VTEP2
VTEP2 apprend VTEP1
```

étape 4

Les hôtes sont annoncés :

route EVPN type 2

```
MAC/IP → VTEP
```

étape 5

Le kernel installe :

```
MAC → vxlan0 dst VTEP
```

9️⃣ Donc la réponse claire

Les VTEP se découvrent grâce à :

```
advertise-all-vni
```

qui génère une **route EVPN type 3 (IMET)** contenant l'adresse du VTEP.

Les autres VTEP apprennent alors :

```
VNI → VTEP
```

et créent automatiquement les entrées VXLAN correspondantes.

💡 Et ça explique **pourquoi EVPN peut supporter des centaines de VTEP sans configuration statique**.

Voici un **morceau de documentation court et clair** que tu peux intégrer dans ton rapport pour expliquer cette sortie.

Observation des routes EVPN échangées entre les VTEP

Afin de vérifier les informations EVPN échangées entre les VTEP, il est possible d'utiliser la commande suivante :

```
show bgp l2vpn evpn neighbors 192.168.0.33 routes
```

Dans notre cas, cette commande est exécutée **depuis le VTEP02** afin d'observer les routes EVPN reçues depuis **le VTEP01 (192.168.0.33)**.

Cette commande permet d'afficher les différents types de routes EVPN échangées via BGP.

La sortie de la commande rappelle également la structure des différents types de routes EVPN :

```
EVPN type-1 prefix
EVPN type-2 prefix
EVPN type-3 prefix
EVPN type-4 prefix
EVPN type-5 prefix
```

Dans notre laboratoire, deux types de routes sont principalement observés :

Routes EVPN Type 2

Les routes **type 2** correspondent aux annonces **MAC/IP des hôtes** situés derrière un VTEP.

Par exemple :

```
[2]:[0]:[48]:[10:e9:92:90:54:40]
```

ou

```
[2]:[0]:[48]:[bc:24:11:5a:37:94]
```

Ces routes indiquent que le VTEP distant annonce les adresses MAC des machines présentes derrière lui.
Le champ `Next Hop` indique l'adresse IP du VTEP qui possède ces machines, ici :

```
192.168.0.33
```

Cela signifie que pour joindre ces adresses MAC, le trafic VXLAN devra être envoyé vers ce VTEP.

Route EVPN Type 3 : découverte des VTEP

On observe également la présence d'une route :

```
[3]:[0]:[32]:[192.168.0.33]
```

Cette entrée correspond à une **route EVPN de type 3**, appelée **IMET (Inclusive Multicast Ethernet Tag)**.

Cette route est utilisée pour annoncer l'existence d'un **VTEP participant au VNI**.

Dans cet exemple :

```
[3]:[0]:[32]:[192.168.0.33]
```

indique que **le VTEP ayant l'adresse IP 192.168.0.33 participe au VNI 5000**.

Cette information est envoyée automatiquement par FRRouting grâce à la commande :

```
advertise-all-vni
```

Lorsque cette route est reçue, le VTEP distant apprend qu'un autre VTEP existe pour ce VNI. Le noyau Linux peut alors créer dynamiquement une entrée VXLAN permettant d'envoyer du trafic vers ce VTEP, même si aucune adresse `remote` n'a été configurée dans l'interface VXLAN.

Conclusion

L'observation de cette sortie BGP EVPN montre donc deux mécanismes essentiels du fonctionnement EVPN :

- les **routes de type 3** permettent aux VTEP de **se découvrir automatiquement** dans le réseau VXLAN
- les **routes de type 2** permettent ensuite d'annoncer les **adresses MAC et IP des machines** situées derrière chaque VTEP

Grâce à ce mécanisme, il n'est plus nécessaire de configurer manuellement les VTEP distants dans l'interface VXLAN.
La découverte et la distribution des informations de couche 2 sont assurées dynamiquement par le **plan de contrôle BGP EVPN**.

Ajout d’un 3 eme VTEP

Je vais maintenant ajouter un 3 eme VTEP qui apprendra tout via notre bgp evpn

J’installe le VTEP

J’installe frr et active bgpd

Ensuite je rajoute une interface reseau ens19 qui servira aux clients derriere le VTEP il sera dans le VLAN50 et toujours VXLAN 5000

Son IP dans le VXLAN : 10.10.10.3

root@srvpxe:~# ip link add br0 type bridge

root@srvpxe:~# ip addr add 10.10.10.3/24

Not enough information: "dev" argument is required.

root@srvpxe:~# ip addr add 10.10.10.3/24 dev br0

root@srvpxe:~# ip link set br0 up

root@srvpxe:~#

ENS 19 NE DOIT PAS AVOIR D IP

root@srvpxe:~# ip link set ens19 master br0

root@srvpxe:~# ip link set ens19 up

root@srvpxe:~# ip link add vxlan0 type vxlan id 5000 local 175.50.0.20 dstport 4789 dev ens18 nolearning

Ensuite

root@srvpxe:~# ip link set vxlan0 up

root@srvpxe:~# ip link set vxlan0 master br0

root@srvpxe:~#

Configurer BGP EVPN sur VTEP3

srvpxe# conf t

srvpxe(config)# router bgp 65001

srvpxe(config-router)# bgp router-id 172.50.0.20

srvpxe(config-router)# neighbor 192.168.0.33 remo

remote-as remove-private-AS

srvpxe(config-router)# neighbor 192.168.0.33 remo

remote-as remove-private-AS

srvpxe(config-router)# neighbor 192.168.0.33 remote-as 65001

srvpxe(config-router)# neighbor 192.168.0.33 update-source ens18

srvpxe(config-router)# neighbor 10.0.0.145 remote-as 65001

srvpxe(config-router)# neighbor 10.0.0.145 update-source ens18

srvpxe(config-router)# address-family l2vpn evpn

srvpxe(config-router-af)# neighbor 192.168.0.33 activate

srvpxe(config-router-af)# neighbor 10.0.0.145 activate

srvpxe(config-router-af)# advertise-all-vni

srvpxe(config-router-af)# exit-address-family

srvpxe(config-router)# end

srvpxe# write

Note: this version of vtysh never writes vtysh.conf

Building Configuration...

Integrated configuration saved to /etc/frr/frr.conf

[OK]

Des deux cotés VTEP 01 et 02 il faut rajouter le nouveau neighbor

Sur VTEP01 :

grogu(config)# router bgp 65001

grogu(config-router)# neighbor 172.50.0.20 remote-as 65001

grogu(config-router)# neighbor 172.50.0.20 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 172.50.0.20 activate

grogu(config-router-af)# end

Ce que nous observons avant de faire la configuration sur le VTEP02

Que entre VTEP01 et VTEP03 l'echange BGP et L2VPN fonctionne parfaitement

```
IPv4 Unicast Summary:
BGP router identifier 172.50.0.20, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 2, using 47 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.145      4      65001         0         4        0    0    0    never       Active        0 N/A
192.168.0.33    4      65001        16        18        0    0    0 00:00:21            0        0 FRRouting/10.3

Total number of neighbors 2

L2VPN EVPN Summary:
BGP router identifier 172.50.0.20, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 3, using 384 bytes of memory
Peers 2, using 47 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.145      4      65001         0         4        0    0    0    never       Active        0 N/A
192.168.0.33    4      65001        16        18       21    0    0 00:00:21            1       17 FRRouting/10.3

Total number of neighbors 2
srvpxe#
```

Et in voit bien que avec le VTEP02 ça ne passe pas encore mais c'est normal

Sur VTEP 02 : grogu# conf t

grogu(config)# router bgp 65001

grogu(config-router)# neighbor 172.50.0.20 remote-as 65001

grogu(config-router)# neighbor 172.50.0.20 update-source ens18

grogu(config-router)# address-family l2vpn evpn

grogu(config-router-af)# neighbor 172.50.0.20 activate

grogu(config-router-af)# end

grogu# write

Voila le resultat

```
srvpxe# show bgp summary

IPv4 Unicast Summary:
BGP router identifier 172.50.0.20, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 2, using 47 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.145      4      65001        16        19        0    0    0 00:00:21            0        0 FRRouting/10.3
192.168.0.33    4      65001        19        23        0    0    0 00:03:55            0        0 FRRouting/10.3

Total number of neighbors 2

L2VPN EVPN Summary:
BGP router identifier 172.50.0.20, local AS number 65001 VRF default vrf-id 0
BGP table version 0
RIB entries 5, using 640 bytes of memory
Peers 2, using 47 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd   PfxSnt Desc
10.0.0.145      4      65001        16        19       23    0    0 00:00:21            3       15 FRRouting/10.3
192.168.0.33    4      65001        19        23       23    0    0 00:03:55            1       15 FRRouting/10.3

Total number of neighbors 2
srvpxe#
```

On voit aussi bien qu'une route de type 3 a été communiqué

```
grogu# show bgp l2vpn evpn route type 3
BGP table version is 251, local router ID is 192.168.0.33
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[EthTag]:[ESI]:[IPlen]:[VTEP-IP]:[Frag-id]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

   Network          Next Hop            Metric LocPrf Weight Path
                    Extended Community
Route Distinguisher: 10.0.0.145:2
 *>i [3]:[0]:[32]:[10.0.0.145]
                    10.0.0.145                    100      0 i
                    RT:65001:5000 ET:8
Route Distinguisher: 172.50.0.20:2
 *>i [3]:[0]:[32]:[175.50.0.20]
                    175.50.0.20                   100      0 i
                    RT:65001:5000 ET:8
Route Distinguisher: 192.168.0.33:2
 *>  [3]:[0]:[32]:[192.168.0.33]
                    192.168.0.33                       32768 i
                    ET:8 RT:65001:5000

Displayed 3 prefixes (3 paths) (of requested type)
grogu#
```

Rappelons que la route de type 3 sert à annoncer les VTEP

C'est le nouveau VTEP qui envoie via une connexion tcp aux voisins de son AS

On voit aussi que les adresses mac ont bien été transmises

```
grogu# show evpn mac vni 5000
Number of MACs (local and remote) known for this VNI: 17
Flags: N=sync-neighs, I=local-inactive, P=peer-active, X=peer-proxy
MAC                Type   Flags Intf/Remote ES/VTEP            VLAN  Seq #'s
00:11:32:c7:10:dc remote       175.50.0.20                           0/0
10:e9:92:90:54:40 remote       175.50.0.20                           0/0
30:5a:3a:07:9c:ea remote       175.50.0.20                           0/0
c8:4a:a0:02:1b:cc remote       175.50.0.20                           0/0
bc:24:11:79:43:a4 remote       10.0.0.145                            0/0
d0:37:45:fd:39:1a remote       175.50.0.20                           0/0
96:12:a6:1a:99:3c remote       175.50.0.20                           0/0
bc:24:11:15:72:11 remote       175.50.0.20                           0/0
64:4e:d7:fd:e9:d1 remote       175.50.0.20                           0/0
bc:24:11:d3:4b:86 remote       10.0.0.145                            0/0
bc:24:11:4c:2e:85 remote       175.50.0.20                           0/0
26:2e:0c:ea:d4:ca remote       175.50.0.20                           0/0
bc:24:11:c7:fb:75 remote       10.0.0.145                            0/0
5e:89:38:5d:91:b0 remote       175.50.0.20                           0/0
bc:24:11:5a:37:94 remote       175.50.0.20                           0/0
bc:24:11:ed:07:42 remote       175.50.0.20                           0/0
b8:d8:2d:47:17:a4 remote       175.50.0.20                           0/0
grogu#
```

Mise en place d’un routeur Underlay dédié (Spine)

Dans notre laboratoire actuel, les VTEP communiquent entre eux à travers le réseau existant de l’infrastructure domestique. Concrètement, le trafic IP servant au transport du VXLAN (le réseau **underlay**) transite par le routeur **OPNsense** présent dans l’environnement réseau de la maison.

Schéma simplifié de la situation initiale :

```
      OPNsense
(routeur domestique)
    /         \
 VTEP1       VTEP2
```

Dans cette configuration, lorsqu’un VTEP souhaite envoyer un paquet VXLAN vers un autre VTEP, le paquet IP est routé par OPNsense avant d’atteindre sa destination. Bien que cela fonctionne correctement, cette approche mélange deux rôles différents :

- le **routage domestique / accès réseau**
- le **transport interne du fabric VXLAN**

Dans un environnement datacenter réel, ces deux fonctions sont généralement **séparées**.

Objectif de la modification

L’objectif est donc de créer une **machine Linux dédiée jouant le rôle de routeur Underlay**, afin de séparer clairement :

- le **réseau de laboratoire EVPN/VXLAN**
- le **réseau domestique et l'accès externe**

Nous allons donc introduire une nouvelle machine Linux qui jouera le rôle de **routeur central** entre les réseaux IP des VTEP.

Le nouveau schéma devient :

```
        Routeur Linux
          (Spine)
          /     \
    VTEP1       VTEP2
```

Dans cette architecture :

- les VTEP n'utilisent plus le routeur domestique pour communiquer entre eux
- ils utilisent **ce routeur central dédié**
- le trafic VXLAN reste entièrement dans l'environnement du laboratoire

---

Principe théorique

Dans une architecture VXLAN/EVPN, il est important de distinguer deux types de réseaux :

Underlay

L'**underlay** est le réseau IP physique qui permet aux VTEP de se joindre.

Il sert uniquement à transporter :

- le trafic VXLAN encapsulé
- les échanges BGP EVPN
- les paquets IP entre VTEP

Les équipements de l'underlay ne voient pas les machines virtuelles ni les réseaux overlay. Ils transportent simplement des paquets IP.

Overlay

L'**overlay** correspond au réseau virtuel créé par VXLAN.

Dans notre laboratoire, ce réseau overlay est par exemple :

```
VNI 5000
Réseau 10.10.10.0/24
```

Les machines connectées derrière les VTEP ont l'impression d'être dans le même réseau local, même si elles sont physiquement situées sur des hôtes différents.

Rôle du routeur Linux (Spine)

La machine Linux introduite dans cette étape joue le rôle d'un **routeur central de l'underlay**.

Son rôle est très simple :

- recevoir les paquets IP provenant d'un VTEP
- déterminer la destination IP
- router le paquet vers le VTEP correspondant

Il ne participe pas au VXLAN lui-même et ne manipule pas les MAC des machines virtuelles. Il transporte simplement les paquets IP encapsulant le VXLAN.

Par exemple, lorsqu'une machine virtuelle située derrière **VTEP1** envoie du trafic vers une machine située derrière **VTEP2**, le chemin suivi est :

```
VM1 → VTEP1 → Routeur Linux → VTEP2 → VM2
```

Le routeur Linux ne voit qu'un paquet IP UDP (port 4789) et se contente de le router.

Intérêt architectural

Introduire ce routeur dédié présente plusieurs avantages :

- **séparation claire du laboratoire et du réseau domestique**
- meilleure compréhension de la distinction **underlay / overlay**
- reproduction plus fidèle des architectures utilisées dans les datacenters modernes

Dans les infrastructures réelles, ce rôle est généralement assuré par des équipements appelés **spines**, qui constituent le cœur du réseau IP reliant les équipements **leaf** (les switches ou hôtes connectant les serveurs).

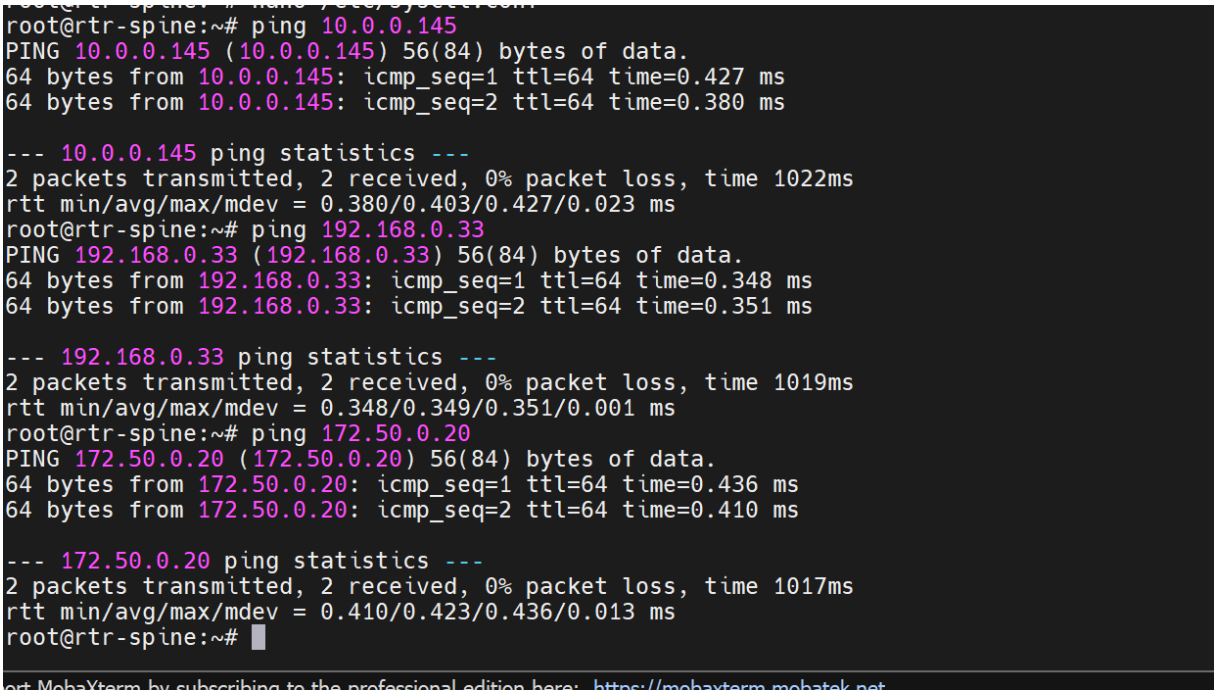
Notre routeur Linux agit donc comme un **spine simplifié**, permettant de structurer l'architecture du laboratoire de manière plus réaliste.

J’installe le serveur routeur

Il aura une patte dans chaque VLAN auxquelles apprtiennent mes VTEP sois le VLAN 100 , 50 et 1

Activer le routage via le fichier sysctl.conf

Je peux pinger les trois vtep



inet 192.168.0.38/24 brd 192.168.0.255 scope global dynamic noprefixroute ens18

inet 172.50.0.23/24 brd 172.50.0.255 scope global dynamic ens18

inet 172.50.0.22/24 brd 172.50.0.255 scope global dynamic ens19

inet 10.0.0.128/24 brd 10.0.0.255 scope global dynamic ens20

Supprimer gateway default et utiliser routeur spine

Sur chaque VTEP on va supprimer la gateway par default et définir le routeur spine comme gateway pour atteindre les réseaux des VTEP les réseaux underlay biensur

Le routeur spine lui ne connait pas l'overlay

Ip route del default ensuite

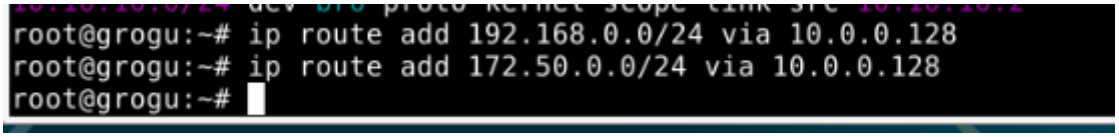
```
root@grogu:~# ip route add 10.0.0.0/24 via 192.168.0.38
root@grogu:~# ping 10.0.0.145
PING 10.0.0.145 (10.0.0.145) 56(84) bytes of data.
64 bytes from 10.0.0.145: icmp_seq=1 ttl=63 time=0.586 ms
64 bytes from 10.0.0.145: icmp_seq=2 ttl=63 time=0.483 ms

--- 10.0.0.145 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.483/0.534/0.586/0.051 ms
root@grogu:~#
```

Et puis pour accéder à VTEP 03

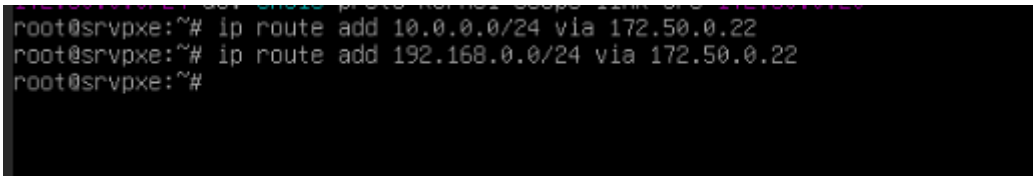
ip route add 172.50.0.0/24 via 192.168.0.38

```
root@grogu:~# ip route add 172.50.0.0/24 via 192.168.0.38
root@grogu:~# ping 172.50.0.20
PING 172.50.0.20 (172.50.0.20) 56(84) bytes of data.
64 bytes from 172.50.0.20: icmp_seq=1 ttl=63 time=0.596 ms
64 bytes from 172.50.0.20: icmp_seq=2 ttl=63 time=0.477 ms
```

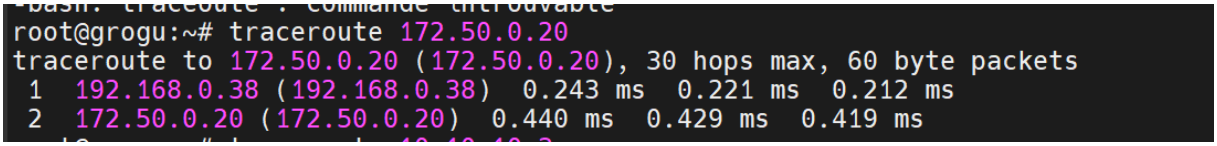
Sur VTEP2



Et enfin sur VTEP 3



Traceroute VTEP 1 vers VTEP3



Ça passe bien par le routeur spine

Voici un **morceau de documentation clair et propre** que tu peux insérer directement dans ton rapport.

Introduction d'un Route Reflector dans l'architecture EVPN

Dans la première partie du laboratoire, les VTEP (VXLAN Tunnel Endpoints) ont été configurés pour établir des sessions **BGP EVPN directement entre eux**. Cette approche fonctionne correctement dans un environnement simple comprenant seulement deux VTEP.

La topologie initiale était donc la suivante :

```
VTEP1  --------  VTEP2
        BGP EVPN
```

Dans cette configuration, chaque VTEP établit une session BGP avec les autres VTEP afin d'échanger les informations nécessaires au fonctionnement de l'overlay VXLAN, notamment :

- les adresses **MAC des machines virtuelles**
- les **adresses IP associées**
- les informations relatives aux **VNI**
- l'adresse IP du **VTEP propriétaire**

Ces informations sont propagées via différents types de routes EVPN (principalement **Type-2 pour les MAC/IP** et **Type-3 pour les VTEP**).

Limites de la topologie en full-mesh

Lorsque le nombre de VTEP augmente, cette architecture devient difficile à maintenir. En effet, BGP nécessite alors une **topologie full-mesh**, c'est-à-dire que chaque VTEP doit établir une session BGP avec tous les autres VTEP.

Le nombre de sessions BGP nécessaires augmente rapidement selon la formule suivante :

[
n(n-1)/2
]

où **n** représente le nombre de VTEP.

Par exemple :

Nombre de VTEP	Sessions BGP nécessaires
2	1
3	3
5	10
10	45

Dans une infrastructure de datacenter comportant plusieurs dizaines ou centaines de VTEP, cette approche devient rapidement **complexe à gérer et peu scalable**.

Introduction du Route Reflector

Pour résoudre ce problème, les architectures VXLAN EVPN utilisent généralement un **Route Reflector (RR)**.

Le principe consiste à introduire un nœud central chargé de recevoir les routes BGP EVPN des VTEP et de les **réfléchir (reflect)** vers les autres VTEP.

Dans notre laboratoire, la machine Linux jouant le rôle de **spine underlay** sera également utilisée comme **Route Reflector EVPN**.

La topologie devient alors :

```
        Spine / Route Reflector
        /          |          \
   VTEP1        VTEP2       VTEP3
```

Chaque VTEP établit uniquement **une session BGP avec le Route Reflector**, ce qui permet de réduire drastiquement le nombre total de sessions BGP nécessaires.

Le Route Reflector :

1. reçoit les routes EVPN annoncées par un VTEP
2. les enregistre dans sa table BGP
3. les redistribue aux autres VTEP clients

Ainsi, les VTEP peuvent continuer à apprendre :

- les **MAC distantes**
- les **IP associées**
- les **VTEP responsables de ces machines**

sans avoir besoin d'établir une session BGP directe entre eux.

Intérêt dans les architectures datacenter

Ce modèle est largement utilisé dans les infrastructures modernes reposant sur EVPN/VXLAN. Les équipements **spine** jouent souvent le rôle de **Route Reflector**, tandis que les équipements **leaf** correspondent aux VTEP connectant les serveurs ou machines virtuelles.

Cette architecture permet :

- une **meilleure scalabilité**
- une **simplification de la configuration BGP**
- une **réduction du nombre de sessions BGP nécessaires**

Dans la suite du laboratoire, le **spine Linux sera configuré comme Route Reflector EVPN**, permettant aux différents VTEP du lab d'échanger leurs routes EVPN via ce nœud central.

Sur le SPINE il faut installer frr

Grossomodo on va sur les VTEP enlever les voisins bgp « VTEP » et ne garder que le routeur spine et le routeur spine recevra les routes de type 2 et 3 de tout les VTEP et les communiquera entre eux comme il sera le neighbor de tous mais les VTEP entre eux ne seront plus des neighbor donc ne recevront plus de transmission de table de routage ou mac vie l2vpn

D'abord sur le routeur spine intégrer l'AS et définir tout les VTEP comme neighbor et activer l2vpn

Sur le routeur spine la partie BGP

```
rtr-spine# conf t
rtr-spine(config)# bgp router 65001
% Unknown command: bgp router 65001
rtr-spine(config)# router bgp 65001
rtr-spine(config-router)# neighbor 172.50.0.20 remote-as 65001
rtr-spine(config-router)# neighbor 172.50.0.20  update-source ens19
rtr-spine(config-router)# neighbor 10.0.0.145 remote-as 65001
rtr-spine(config-router)# neighbor 10.0.0.145 update-source ens20
rtr-spine(config-router)# neighbor 192.168.0.33 remote-as 65001
rtr-spine(config-router)# neighbor 192.168.0.33 update-source ens18
```

Activer EVPN

```
rtr-spine(config-router)# address-family l2vpn evpn
rtr-spine(config-router-af)# neighbor 172.50.0.20 activate
rtr-spine(config-router-af)# neighbor 10.0.0.145 activate
rtr-spine(config-router-af)# neighbor 192.168.0.33 activate
```

Activer le route reflector

```
rtr-spine(config-router-af)# neighbor 172.50.0.20 route-reflector-client
rtr-spine(config-router-af)# neighbor 10.0.0.145 route-reflector-client
rtr-spine(config-router-af)# neighbor 192.168.0.33 route-reflector-client
rtr-spine(config-router-af)#
```

Configuration des VTEP

VTEP3

Supprimer les neighbor dans BGP

```
srvpxe(config)# router bgp 65001
srvpxe(config-router)# no neighbor 1
10.0.0.145    192.168.0.33
srvpxe(config-router)# no neighbor 10.0.0.145
srvpxe(config-router)# no neighbor 192.168.0.33
srvpxe(config-router)#
```

Ajout du routeur spine en eighbor

```
srvpxe(config-router)# neighbor 172.50.0.22 remote-as 65001
srvpxe(config-router)# neighbor 172.50.0.22 activate
srvpxe(config-router)# address-family l2vpn evpn
srvpxe(config-router-af)# neighbor 172.50.0.22 activate
srvpxe(config-router-af)# advertise-all-vni
srvpxe(config-router-af)#
```

Et on fait pareil sur tout les VTEP

Voici une **version propre, complète et pédagogique** que tu peux intégrer directement dans ta documentation du lab. J'ai tout repris depuis le début du problème et ajouté **l'étape importante sur la MAC du bridge**.

Dépannage EVPN après introduction d'un Route Reflector (Spine)

1. Contexte du lab

Au départ, l'architecture VXLAN EVPN fonctionnait avec une **topologie full-mesh BGP entre les VTEP**.

Chaque VTEP échangeait directement les routes EVPN avec les autres.

Architecture initiale :

```
VTEP1 -------- BGP -------- VTEP2
192.168.0.33               10.0.0.145
```

Dans cette configuration :

- chaque VTEP connaît directement les autres VTEP
- les routes EVPN (MAC/IP) sont échangées directement
- le VXLAN fonctionne correctement

---

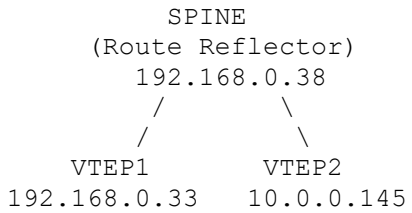
2. Évolution de l'architecture

Afin de se rapprocher d'une architecture **datacenter réelle**, un **routeur Spine** a été introduit.

Ce routeur joue le rôle de :

- **Route Reflector BGP**
- point central d'échange des routes EVPN

Nouvelle topologie :



Dans cette architecture :

- les VTEP **ne sont plus voisins entre eux**
- ils ne parlent **qu'au routeur Spine**
- le Spine redistribue les routes EVPN

C'est une architecture classique **leaf-spine**.

---

3. Problème rencontré

Après suppression des voisins BGP entre les VTEP et configuration du **route reflector**, le comportement suivant a été observé.

Le control plane fonctionnait

Les sessions BGP étaient **UP** :

```
show bgp summary
```

Les routes EVPN étaient échangées correctement :

```
show bgp l2vpn evpn
```

Les MAC distantes apparaissaient correctement.

Mais les pings entre VTEP ne passaient plus

Malgré un control plane EVPN fonctionnel, le dataplane VXLAN ne fonctionnait plus :

```
ping 10.10.10.2
```

échouait.

Cela indique que :

Plan	État
Control plane EVPN	OK
Data plane VXLAN	KO

---

4. Analyse du problème

Pour analyser le problème, la table ARP EVPN a été vérifiée :

```
vtysh -c "show evpn arp-cache vni 5000"
```

Résultat observé :

```
10.10.10.2 local active c2:75:36:2d:80:a3
```

Ce résultat est incorrect.

L'adresse **10.10.10.2 appartient au VTEP2**, donc elle devrait apparaître comme :

```
remote
```

et non **local**.

Cela signifie que le VTEP1 pensait que **10.10.10.2 était associée à sa propre MAC**.

---

5. Causes identifiées

Deux problèmes principaux ont été identifiés.

Problème 1 : IP présente sur une interface du bridge

L'interface `ens19` appartenait au bridge `br0`, mais possédait encore une adresse IP.

Exemple observé :

```
ens19 192.168.0.40/24
```

Dans une architecture VXLAN EVPN :

- les interfaces physiques du bridge doivent être **L2 uniquement**
- les IP doivent être configurées **sur le bridge br0**

Sinon le bridge peut apprendre des informations incohérentes.

Correction appliquée :

```
ip addr flush dev ens19
```

Problème 2 : MAC identique sur les bridges des VTEP

Après redémarrage de l'infrastructure, les deux bridges `br0` possédaient **la même adresse MAC**.

Cela a provoqué une confusion dans l'apprentissage ARP EVPN.

Exemple :

```
VTEP1 br0 MAC = c2:75:36:2d:80:a3
VTEP2 br0 MAC = c2:75:36:2d:80:a3
```

Le VTEP1 associait donc l'adresse IP distante à **sa propre MAC**, ce qui empêchait la résolution correcte.

Correction appliquée

Une MAC différente a été attribuée aux bridges.

Sur VTEP1

```
ip link set dev br0 address 02:00:00:00:10:01
```

Sur VTEP2

```
ip link set dev br0 address 02:00:00:00:10:02
```

Ces MAC étant maintenant différentes, l'apprentissage ARP EVPN fonctionne correctement.

---

6. Vérification du VXLAN

La configuration VXLAN utilisée est la suivante :

```
ip link add vxlan0 type vxlan id 5000 \
local 192.168.0.33 \
dev ens18 \
dstport 4789 \
nolearning
```

Puis :

```
ip link set vxlan0 master br0
ip link set vxlan0 up
```

Signification des paramètres :

Paramètre	Rôle
VNI	identifiant du réseau VXLAN
local	IP du VTEP

Paramètre	Rôle
dev	interface underlay
dstport	port VXLAN
nolearning	apprentissage MAC contrôlé par EVPN

---

7. Vérification des MAC EVPN

Commande utilisée :

```
show evpn mac vni 5000
```

Exemple :

```
MAC               Type   Remote VTEP
bc:24:11:79:43:a4 remote 10.0.0.145
```

Cela confirme que les MAC distantes sont bien apprises via EVPN.

---

8. Vérification de la table FDB Linux

Commande :

```
bridge fdb show
```

Exemple :

```
bc:24:11:79:43:a4 dev vxlan0 dst 10.0.0.145 extern_learn
```

Signification :

Élément	Rôle
extern_learn	MAC apprise par EVPN
dst	VTEP distant

---

9. Nettoyage de la table ARP

Après corrections :

```
ip neigh flush all
```

---

10. Résultat final

Après correction :

```
show evpn arp-cache vni 5000
```

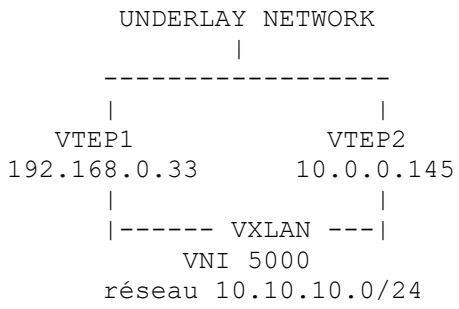
donne :

```
10.10.10.2 remote active <MAC> 10.0.0.145
```

Les pings fonctionnent désormais :

```
ping 10.10.10.2
```

Architecture finale



Les routes EVPN sont échangées via le **Spine (Route Reflector)**.

Points importants à retenir

1️⃣ Les interfaces physiques du bridge doivent être L2 uniquement

```
ip addr flush dev ens19
```

2️⃣ L'adresse IP doit être configurée sur le bridge

```
br0
```

3 Les MAC des bridges doivent être uniques

Sinon EVPN peut associer les IP distantes à une MAC locale.

---

4 Commandes essentielles de debug

```
show bgp l2vpn evpn
show evpn mac vni 5000
show evpn arp-cache vni 5000
bridge fdb show
```

Conclusion

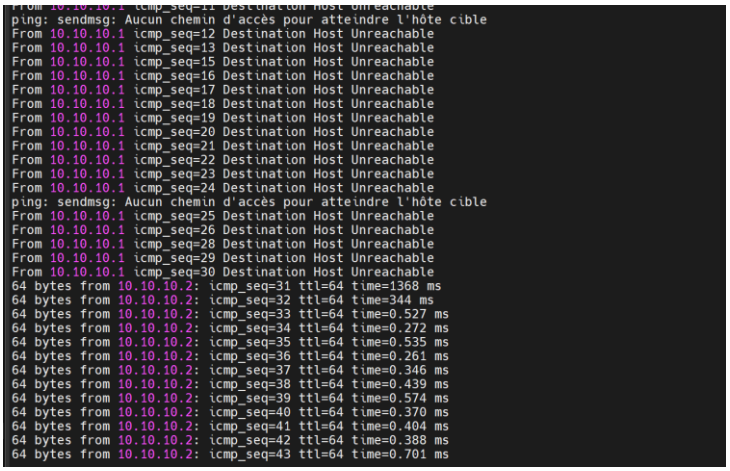
Le problème rencontré n'était **pas lié au route reflector ni au BGP EVPN**, mais à une incohérence locale dans la configuration du bridge.

Deux erreurs ont provoqué le dysfonctionnement :

1. une interface du bridge possédait encore une IP
2. les bridges des VTEP possédaient la même adresse MAC

Après correction de ces deux points, l'apprentissage ARP EVPN et le dataplane VXLAN ont recommencé à fonctionner correctement.

J'ai lancé un ping et en meme temps j'ai deceonnecté le routeur spine on vout qu'il passe pas et des que je rebranche le routeur spine au reseau le ping passe



Rajout d’un second VNI

Pour bien tout explorer dans notre lab nous allons rajouter un second VNI car en prod il y en a rarement qu’un seul