REPUBLIQUE DU SENEGAL

UN PEUPLE-UN BUT-UNE FOI

EC2LT
ECOLE CENTRALE DES LOGICIELS LIBRES
ET DE TELECOMMUNICATIONS

RAPPORT

SDN et VxLAN

RESEAUX ET TELECOMMUNICATIONS

OPTION : CYBERSECURITE/DEVOPS

Rédiger par :

Serge Elvis Espoir BOUNGUELE

Sous la direction de :

Pr Samuel OUYA

Année Académique 2024-2025

INTRODUCTION GENERALE

Dans un contexte de croissance exponentielle des besoins en connectivité et d'évolution rapide des infrastructures réseau, les technologies telles que le **VXLAN (Virtual Extensible LAN)** et le **Software-Defined Networking (SDN)** jouent un rôle central dans la transformation des datacenters et des réseaux d'entreprise. Le **VXLAN** répond aux limitations des réseaux locaux traditionnels (VLAN), en offrant une plus grande évolutivité avec la capacité de créer jusqu'à 16 millions de réseaux logiques superposés sur une infrastructure IP. Cela permet aux entreprises de surmonter les contraintes des VLANs classiques, notamment dans les environnements cloud et multi-tenant où un grand nombre de machines virtuelles et de réseaux isolés doivent coexister.

D'un autre côté, le **SDN** révolutionne la gestion et l'optimisation des réseaux en introduisant une séparation entre le plan de contrôle et le plan de données. Cette approche permet une gestion centralisée, automatisée et programmable du réseau, offrant une flexibilité et une agilité accrues dans la réponse aux besoins des entreprises. Le SDN s'adapte parfaitement aux environnements modernes où les demandes en connectivité évoluent rapidement, que ce soit avec l'émergence du cloud computing, de l'Internet des objets (IoT) ou des réseaux 5G.

Dès lors, VXLAN et SDN apportent une solution complète pour répondre aux défis croissants en matière de scalabilité, d'efficacité et de gestion des réseaux, en permettant aux entreprises de créer des infrastructures plus agiles, sécurisées et faciles à gérer.

SOMMAIRE

PARIE I : INTRODUCTION AU SDN (SOFTWARE-DEFINED NETWORKING)

Le **Software-Defined Networking (SDN)**, ou réseau défini par logiciel, est une approche révolutionnaire pour la gestion et l'optimisation des réseaux informatiques. Contrairement aux réseaux traditionnels, où les fonctions de contrôle et de transmission sont intimement liées, le SDN introduit une séparation claire entre ces deux plans. Le plan de contrôle est extrait et centralisé, tandis que le plan de données continue de transmettre le trafic, mais sous la direction d'un contrôleur SDN central. Cette architecture innovante permet une gestion plus souple, automatisée et programmable du réseau, réduisant la complexité opérationnelle tout en améliorant l'agilité et la réactivité des infrastructures réseau.

Le SDN est un élément clé pour faire face aux défis posés par l'évolution rapide des besoins en connectivité et par l'essor des technologies comme le cloud computing, l'Internet des objets (IoT) et les réseaux 5G. Il simplifie la configuration et l'administration des réseaux, tout en permettant aux organisations de réagir rapidement aux demandes en constante évolution.

I.1 Séparation du plan de contrôle et du plan de données

Dans un réseau **SDN**, le plan de contrôle (responsable de la prise de décisions sur le routage des paquets) est séparé du plan de données (responsable de la transmission réelle des paquets). Le plan de contrôle est centralisé dans un contrôleur SDN qui gère le comportement global du réseau.

I.2 Contrôleur SDN

Le contrôleur SDN est le cerveau du réseau SDN. Il centralise la gestion du réseau et décide comment les paquets de données doivent être acheminés. Grâce à son contrôle global, il peut optimiser la performance et la sécurité du réseau.

I.3 Programmabilité

Un autre avantage clé du SDN est la programmabilité. Les administrateurs peuvent écrire des programmes pour gérer dynamiquement les ressources réseau, ce qui permet d'automatiser les processus et de répondre rapidement aux changements dans les besoins de l'entreprise.

I.4 Interopérabilité et virtualisation

Le SDN favorise l'utilisation d'API ouvertes et la compatibilité avec diverses technologies, permettant ainsi aux infrastructures réseau d'être plus flexibles. Il facilite également la virtualisation des réseaux (**Network Function Virtualization - NFV**), permettant la création de réseaux virtuels fonctionnant indépendamment de l'infrastructure matérielle sous-jacente.

I.5 Applications et bénéfices du SDN

Le SDN offre plusieurs avantages pratiques :

- **Optimisation des coûts** : il permet de réduire les coûts opérationnels en centralisant la gestion et en diminuant la nécessité d'interventions manuelles.
- **Flexibilité et agilité** : grâce à la programmabilité, le SDN permet aux réseaux de s'adapter rapidement aux nouvelles exigences de l'entreprise, comme le déploiement rapide d'applications ou l'ajustement des capacités.
- **Sécurité améliorée** : le SDN offre des mécanismes avancés de gestion de la sécurité, en permettant l'application centralisée de politiques de sécurité et en détectant plus rapidement les anomalies.
- **Automatisation et orchestration** : avec le SDN, des tâches auparavant manuelles et complexes, telles que la configuration du routage ou l'optimisation de la bande passante, peuvent être automatisées.

TP1 : FONCTIONNEMENT D'OPENSWITCH ET QUELQUES COMMANDES A MAITRISER POUR GERER LES SWITCHES OPENFLOW : OVS-VSCTL, OVS-OFCTL, OVS-DPCTL

1. **Mise à jour du système**

root@espoir:/home/espoir# sudo apt update

```
root@espoir:/home/espoir# sudo apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Get:2 https://deb.nodesource.com/node_20.x nodistro InRelease [12,1 kB]
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:4 https://deb.nodesource.com/node_20.x nodistro/main amd64 Packages [9 154 B]
Get:5 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
```

```
root@espoir:/home/espoir# sudo apt upgrade -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-107-generic linux-hwe-5.15-headers-5.15.0-107
  linux-image-5.15.0-107-generic linux-modules-5.15.0-107-generic
```

2. Installation d'Open vSwitch

root@espoir:/home/espoir# sudo apt install openvswitch-switch -y

```
root@espoir:/home/espoir# sudo apt install openvswitch-switch -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-107-generic linux-hwe-5.15-headers-5.15.0-107
  linux-image-5.15.0-107-generic linux-modules-5.15.0-107-generic
  linux-modules-extra-5.15.0-107-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libevent-2.1-7 libunbound8 openvswitch-common python3-openvswitch
  python3-sortedcontainers
```

3. Vérification de l'installation

Après l'installation, vérifions que le service **Open vSwitch** est actif et fonctionne :

root@espoir:/home/espoir# sudo systemctl status openvswitch-switch

```
root@espoir:/home/espoir# sudo systemctl status openvswitch-switch
● openvswitch-switch.service - Open vSwitch
     Loaded: loaded (/lib/systemd/system/openvswitch-switch.service; enabled; ve>
     Active: active (exited) since Thu 2024-08-29 01:57:24 GMT; 3min 7s ago
   Main PID: 21296 (code=exited, status=0/SUCCESS)
      Tasks: 0 (limit: 4540)
     Memory: 0B
     CGroup: /system.slice/openvswitch-switch.service

août 29 01:57:24 espoir systemd[1]: Starting Open vSwitch...
août 29 01:57:24 espoir systemd[1]: Finished Open vSwitch.
lines 1-10/10 (END)
```

4. Démarrage automatique au démarrage du système

Si nous souhaitons que **Open vSwitch** démarre automatiquement avec le système :

root@espoir:/home/espoir# sudo systemctl enable openvswitch-switch

```
root@espoir:/home/espoir# sudo systemctl enable openvswitch-switch
Synchronizing state of openvswitch-switch.service with SysV service script with /
lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable openvswitch-switch
root@espoir:/home/espoir#
```

La commande **ovs-vsctl show** affiche un résumé de la configuration de tous les commutateurs.

root@espoir:/home/espoir# sudo ovs-vsctl show

```
root@espoir:/home/espoir# sudo ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    ovs_version: "2.13.8"
root@espoir:/home/espoir#
```

5.Gestion de la configuration Open vSwitch

La commande **ovs-vsctl** est utilisée pour gérer la configuration des commutateurs (switches) virtuels d'**Open vSwitch**.

Affichage de la version d'**Open vSwitch**

root@espoir:/home/espoir# ovs-vsctl -V

```
root@espoir:/home/espoir# ovs-vsctl -V
ovs-vsctl (Open vSwitch) 2.13.8
DB Schema 8.2.0
root@espoir:/home/espoir#
```

Cette commande affiche la **version 2.13.8** actuelle d'Open vSwitch.

6. Ajout d'un commutateur

La commande **ovs-vsctl add-br switch-noir** crée un nouveau commutateur appelé switch-noir.

root@espoir:/home/espoir# ovs-vsctl add-br switch-noir

```
root@espoir:/home/espoir# ovs-vsctl add-br switch-noir
root@espoir:/home/espoir#
```

7. Ajout d’un port à un commutateur

```
root@espoir:/home/espoir# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20c:29ff:fea7:5626  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a7:56:26  txqueuelen 1000  (Ethernet)
        RX packets 200800  bytes 302443623 (302.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 50445  bytes 3296200 (3.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.13  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::7929:ae2b:5b15:5247  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a7:56:30  txqueuelen 1000  (Ethernet)
        RX packets 2764  bytes 315580 (315.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1544  bytes 148733 (148.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

root@espoir:/home/espoir# ovs-vsctl add-port switch-noir ens33

Cela ajoute l'interface réseau **ens33** comme port au commutateur **switch-noir**.

```
root@espoir:/home/espoir# ovs-vsctl add-port switch-noir ens33
root@espoir:/home/espoir#
```

8. Configuration du contrôleur pour le commutateur

root@espoir:/home/espoir# ovs-vsctl set-controller switch-noir tcp:127.0.0.1:6633

```
root@espoir:/home/espoir# ovs-vsctl set-controller switch-noir tcp:127.0.0.1:6633
root@espoir:/home/espoir#
```

Cette commande configure **switch-noir** pour qu’il soit contrôlé par un contrôleur à l'adresse IP **127.0.0.1** et au port **6633**.

9. Gestion des flux OpenFlow avec la commande ovs-ofctl

Cette commande permet de gérer les flux de données au sein des commutateurs **OpenFlow**.

- **Ajout d’un flux pour traiter le trafic normalement**

root@espoir:/home/espoir# sudo ovs-ofctl add-flow switch-noir actions=NORMAL

```
root@espoir:/home/espoir# sudo ovs-ofctl add-flow switch-noir actions=NORMAL
root@espoir:/home/espoir#
```

Cela configure **switch-noir** pour traiter les paquets comme un commutateur standard.

- **Suppression de tous les flux d'un commutateur**

root@espoir:/home/espoir# sudo ovs-ofctl del-flows switch-noir

Cette commande supprime tous les flux configurés sur **switch-noir**.

```
root@espoir:/home/espoir# sudo ovs-ofctl del-flows switch-noir
root@espoir:/home/espoir#
```

Ajout d'un flux pour rediriger le trafic

La commande suivante redirige le trafic entrant sur le **port 1** vers le **port 2** de **switch-noir**.

root@espoir:/home/espoir# sudo ovs-ofctl add-flow switch-noir in_port=1,actions=output:2

```
root@espoir:/home/espoir# sudo ovs-ofctl add-flow switch-noir in_port=1,actions=output:2
root@espoir:/home/espoir#
```

Affichage des détails des ports en utilisant le protocole OpenFlow 1.3

Il y'a affichage d'informations détaillées sur les ports de **switch-noir** utilisant **OpenFlow 1.3**.

root@espoir:/home/espoir# sudo ovs-ofctl add-flow switch-noir in_port=1,actions=output:2

```
root@espoir:/home/espoir# sudo ovs-ofctl --protocols=OpenFlow13 dump-ports switch-noir
OFPST_PORT reply (OF1.3) (xid=0x2): 2 ports
  port LOCAL: rx pkts=782, bytes=55036, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=36, bytes=6040, drop=0, errs=0, coll=0
           duration=1863.127s
  port  ens33: rx pkts=201988, bytes=302560593, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=50461, bytes=3297772, drop=0, errs=0, coll=0
           duration=1518.035s
root@espoir:/home/espoir#
```

10. Gestion des chemins de données Open vSwitch avec la commande ovs-dpctl

Cette commande est utilisée pour gérer les chemins de données (**data paths**) dans Open vSwitch, en dehors de **ovs-vswitchd**.

Lister les datapaths

Pour voir les **datapaths** actuellement configurés sur notre système :

root@espoir:/home/espoir# sudo ovs-dpctl show

```
root@espoir:/home/espoir# sudo ovs-dpctl show
system@ovs-system:
  lookups: hit:1389 missed:357 lost:0
  flows: 2
  masks: hit:2271 total:2 hit/pkt:1.30
  port 0: ovs-system (internal)
  port 1: switch-noir (internal)
  port 2: ens33
root@espoir:/home/espoir#
```

Ajouter un datapath

Pour créer un nouveau **datapath**, nous pouvons utiliser la commande suivante :

root@espoir:/home/espoir# sudo ovs-dpctl add-dp dp0

```
root@espoir:/home/espoir# sudo ovs-dpctl add-dp dp0
root@espoir:/home/espoir#
```

Cela créera un **datapath** nommé **dp0**.

- **Ajout d'une interface à un datapath**

On créer ou renomme l'interface en utilisant des outils comme **ip** pour créer une interface virtuelle.

root@espoir:/home/espoir# sudo ip link add name eth1 type dummy

root@espoir:/home/espoir# sudo ip link set eth1 up

```
root@espoir:/home/espoir# sudo ip link add name eth1 type dummy
root@espoir:/home/espoir# sudo ip link set eth1 up
root@espoir:/home/espoir#
```

Ensuite, on ajoute une interface à un datapath on effectue cette commande :

root@espoir:/home/espoir# sudo ovs-dpctl add-if dp0 eth1

```
root@espoir:/home/espoir# sudo ovs-dpctl add-if dp0 eth1
root@espoir:/home/espoir#
```

Cela ajoute l'interface **eth1** au **datapath dp0**

- **Verification**

root@espoir:/home/espoir# sudo ovs-dpctl show

```
root@espoir:/home/espoir# sudo ovs-dpctl show
system@ovs-system:
  lookups: hit:2249 missed:592 lost:0
  flows: 4
  masks: hit:3828 total:2 hit/pkt:1.35
  port 0: ovs-system (internal)
  port 1: switch-noir (internal)
  port 2: ens33
system@dp0:
  lookups: hit:0 missed:0 lost:0
  flows: 0
  masks: hit:0 total:0 hit/pkt:0.00
  port 0: dp0 (internal)
  port 1: eth1
```

- **Suppression d'un datapath**

Pour supprimer un **datapath** on execute la commande suivante :

```
root@espoir:/home/espoir# sudo ovs-dpctl del-dp dp0
root@espoir:/home/espoir#
```

Cela supprimera le **datapath dp0.**

- **Verification**

```
root@espoir:/home/espoir# sudo ovs-dpctl show
system@ovs-system:
  lookups: hit:2434 missed:628 lost:0
  flows: 1
  masks: hit:4149 total:1 hit/pkt:1.35
  port 0: ovs-system (internal)
  port 1: switch-noir (internal)
  port 2: ens33
root@espoir:/home/espoir#
```

11. Utilisation de ovsdb-tool

La commande **ovsdb-tool** est utilisé pour gérer les fichiers de base de données **OVSDB**. C'est un outil puissant pour la maintenance et la manipulation des bases de données utilisées par **Open vSwitch**.

- **Création d'une nouvelle base de données**

Pour créer une nouvelle base de données **master1** avec un schéma donné, on utilise la commande suivante :

root@espoir:/home/espoir# sudo ovsdb-tool create /home/espoir/master1.db /usr/share/openvswitch/vswitch.ovsschema

```
root@espoir:/home/espoir# sudo ovsdb-tool create /home/espoir/master1.db /usr/share/openvswitch/vswit
ch.ovsschema
root@espoir:/home/espoir#
```

- **Démarrer un serveur OVS avec votre base de données**

#sudo ovsdb-server remote=punix:/var/run/openvswitch/db.sock --remote=db:Open_vSwitch,Open_vSwitch,Open_vSwitch --pidfile --detach

```
root@espoir:/home/espoir# sudo ovsdb-server --remote=punix:/var/run/openvswitch/db.sock --remote=db:O
pen_vSwitch,Open_vSwitch,Open_vSwitch --pidfile --detach
ovsdb-server: /var/run/openvswitch/ovsdb-server.pid: already running as pid 21154, aborting
root@espoir:/home/espoir#
```

- **Afficher le contenu de la base de données**

Pour afficher le contenu d'une base de données **OVSDB** dans un format lisible, on utilise la commande suivante :

root@espoir:/home/espoir# sudo ovsdb-tool show-log /var/lib/openvswitch/conf.db

```
root@espoir:/home/espoir# sudo ovsdb-tool show-log /var/lib/openvswitch/conf.db
record 0: "Open_vSwitch" schema, version="8.2.0", cksum="1076640191 26427"

record 1: 2024-08-29 01:57:21.549 "ovs-vsctl (invoked by /bin/sh): ovs-vsctl --no-wait -- init -- set
 Open_vSwitch . db-version=8.2.0"
record 2: 2024-08-29 01:57:21.580 "ovs-vsctl (invoked by /bin/sh): ovs-vsctl --no-wait set Open_vSwit
ch . ovs-version=2.13.8 "external-ids:system-id=\"a4ea1a1f-7ed9-43b9-85e0-17d2d863abae\"" "external-i
ds:rundir=\"/var/run/openvswitch\"" "system-type=\"ubuntu\"" "system-version=\"20.04\""
record 3: 2024-08-29 01:57:21.864
record 4: 2024-08-29 01:57:21.865
record 5: 2024-08-29 01:57:21.949 "ovs-vsctl (invoked by /bin/sh): ovs-vsctl --no-wait add Open_vSwit
ch . external-ids hostname=localhost"
record 6: 2024-08-29 02:30:40.345 "ovs-vsctl (invoked by bash): ovs-vsctl add-br switch-noir"
record 7: 2024-08-29 02:30:40.454
record 8: 2024-08-29 02:30:40.458
record 9: 2024-08-29 02:36:25.525 "ovs-vsctl (invoked by bash): ovs-vsctl add-port switch-noir ens33"
record 10: 2024-08-29 02:36:25.533
record 11: 2024-08-29 02:39:57.863 "ovs-vsctl (invoked by bash): ovs-vsctl set-controller switch-noir
 tcp:127.0.0.1:6633"
record 12: 2024-08-29 02:39:57.866
root@espoir:/home/espoir#
```

TP 2: OPENSWITCH ET DOCKER

L'objectif de ce **tp** est de savoir relier un conteneur à un port switch Openvswitch

- **Mise à jour de l'installation des paquets**

```
root@espoir:/home/espoir# apt update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 https://deb.nodesource.com/node_20.x nodistro InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
Hit:5 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 128 kB in 2s (71,0 kB/s)
```

root@espoir:/home/espoir# apt -y install net-tools docker.io openvswitch-switch

Afin de bien procédée, nous allons installer sur la machine les outils comme : **net-tools docker** et **openvswitch.**

```
root@espoir:/home/espoir# apt -y install net-tools docker.io openvswitch-switch
Reading package lists... Done
Building dependency tree
Reading state information... Done
net-tools is already the newest version (1.60+git20180626.aebd88e-1ubuntu1).
openvswitch-switch is already the newest version (2.13.8-0ubuntu1.4).
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-107-generic linux-hwe-5.15-headers-5.15.0-107 linux-image-5.15.0-107-generic
  linux-modules-5.15.0-107-generic linux-modules-extra-5.15.0-107-generic
```

2.1 Téléchargement de l'image ubuntu

root@espoir:/home/espoir# docker pull ubuntu

```
root@espoir:/home/espoir# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
31e907dcc94a: Pull complete
Digest: sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@espoir:/home/espoir#
```

2.2 Verification de l'image

root@espoir:/home/espoir# docker images

```
root@espoir:/home/espoir# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    edbfe74c41f8   3 weeks ago   78.1MB
root@espoir:/home/espoir#
```

Nous allons lancer un conteneur à partir duquel, on créer une nouvelle image **ubuntu1**

root@espoir:/home/espoir# docker run -it ubuntu /bin/bash

```
root@espoir:/home/espoir# docker run -it ubuntu /bin/bash
root@930d0c137b43:/#
```

On installe les paquets net-tools et **iputils-ping** sur le conteneur

root@930d0c137b43:/# apt update

```
root@930d0c137b43:/# apt update
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [12.7 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [354 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [337 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [404 kB]
```

root@930d0c137b43:/# apt install net-tools iputils-ping

```
root@930d0c137b43:/# apt install net-tools iputils-ping
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcap2-bin libpam-cap
The following NEW packages will be installed:
  iputils-ping libcap2-bin libpam-cap net-tools
0 upgraded, 4 newly installed, 0 to remove and 29 not upgraded.
```

Note : nous devrions bien noter l'id du conteneur, dans notre cas c'est : **930d0c137b43**

Dans un autre terminal, on crée une nouvelle image **ubuntu1** à partir de l'id du conteneur précédent.

root@espoir:/home/espoir# docker ps

```
root@espoir:/home/espoir# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
930d0c137b43   ubuntu    "/bin/bash"   16 minutes ago   Up 16 minutes             sweet_einstein
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# docker commit 930d0c137b43 ubuntu1

```
root@espoir:/home/espoir# docker commit 930d0c137b43 ubuntu1
sha256:d908307a2b0d9fbe52bbd4d3db328ff0338075d7e94448fa1949329780d55ef9
root@espoir:/home/espoir#
```

Note : nous devrions savoir que **ubuntu** est une image ayant les utilitaires réseaux **ifconfig, ping**

On créer sur la **machine1 docker1** et **docker2** sans avoir les rattacher à un réseau dans un premier temps.

root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@7595d6fed04e:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@b52a8f59af6e:/#
```

Créons le switch **ovs-br0**

root@espoir:/home/espoir# ovs-vsctl add-br ovs-br0

```
root@espoir:/home/espoir# ovs-vsctl add-br ovs-br0
root@espoir:/home/espoir#
```

Ensuite, créons l'interface interne sur la machine

root@espoir:/home/espoir# ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal

```
root@espoir:/home/espoir# ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal
root@espoir:/home/espoir#
```

Puis donnons l'adresse IP à cette interface et on les active sur la machine

root@espoir:/home/espoir# ip address add 192.168.1.1/24 dev veth0

```
root@espoir:/home/espoir# ip address add 192.168.1.1/24 dev veth0
root@espoir:/home/espoir#
```

Utilisant l'outil **ovs-docker** pour ajouter sur la machine des adresses **IP** aux conteneurs **1** et **2**

root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker1 --ipaddress=192.168.1.11/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker1 --ipaddress=192.168.1.11/24 --gate
way=192.168.1.1
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker2 --ipaddress=192.168.1.12/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker2 --ipaddress=192.168.1.12/24 --gate
way=192.168.1.1
root@espoir:/home/espoir#
```

2.3 Verification des adresse IP sur chaque machine

Pour docker 1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@7595d6fed04e:/#
root@7595d6fed04e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether d6:e1:3f:30:69:0a  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4870 (4.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Pour docker 2

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@b52a8f59af6e:/#
root@b52a8f59af6e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 2a:4d:5e:f0:73:f5  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4755 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Utilisant la commande **ovs-vsctl show** pour afficher une vue d'ensemble de la configuration actuelle des commutateurs **Open vSwitch**. Elle fournit un aperçu détaillé de la base de données de configuration d'**Open vSwitch** (**OVS**).

root@espoir:/home/espoir# sudo ovs-vsctl show

```
root@espoir:/home/espoir# sudo ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge switch-noir
        Controller "tcp:127.0.0.1:6633"
        Port ens33
            Interface ens33
        Port switch-noir
            Interface switch-noir
                type: internal
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port ovs-br0
            Interface ovs-br0
                type: internal
        Port "97dbf2363c114_l"
            Interface "97dbf2363c114_l"
        Port "169681d9c9ef4_l"
            Interface "169681d9c9ef4_l"
    ovs_version: "2.13.8"
root@espoir:/home/espoir#
```

Voici une explication des éléments principaux :

1. **UUID du Bridge**

- **124f7c01-ddc0-4fa4-ad32-9ede69b528eb**: il s'agit d'un identifiant unique (**UUID**) généré par **OVS** pour notre instance de configuration.

2. **Bridge switch-noir**

- **Bridge** : **switch-noir** est le nom d'un bridge configuré.
- **Controller** : **tcp:127.0.0.1:6633** indique qu'un contrôleur est configuré pour gérer ce bridge via l'adresse **IP 127.0.0.1** (localhost) et le port **6633.**
- **Port ens33** : représente un port physique ou virtuel sur le bridge **switch-noir.**
- **Interface ens33** : cette interface est associée au port **ens33**.
- **Port switch-noir** : il y a également un port interne avec le même nom que le bridge.
- **Interface switch-noir** : cette interface est de type **internal**, ce qui signifie qu'elle est utilisée pour la communication interne entre **OVS** et le système d'exploitation.

3. **Bridge ovs-br0**

- **Bridge** : **ovs-br0** est un autre bridge configuré.
- **Port veth0** : représente un port virtuel de type **internal** sur le bridge **ovs-br0**.
- **Interface veth0** : cette interface est de type **internal**.
- **Port ovs-br0** : représente également un port interne avec le même nom que le bridge.
- **Interface ovs-br0** : cette interface est aussi de type **internal**.
- **Port 97dbf2363c114_l et 169681d9c9ef4_l** : ce sont des interfaces virtuelles ou des interfaces réseau associées à des conteneurs.

4. **Version de OVS**

- **ovs_version: "2.13.8"** : indique la version **d'Open vSwitch** installée et en cours d'exécution, qui est la **version 2.13.8**.

Créons notre troisième conteneur nommé **docker 3**

root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1
root@af20cea98808:/#
```

root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker3 --ipaddress=192.168.1.13/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker3 --ipaddress=192.168.1.13/24 --gate
way=192.168.1.1
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# ovs-vsctl show

```
root@espoir:/home/espoir# ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge switch-noir
        Controller "tcp:127.0.0.1:6633"
        Port ens33
            Interface ens33
        Port switch-noir
            Interface switch-noir
                type: internal
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port ovs-br0
            Interface ovs-br0
                type: internal
        Port "73e24986d7514_l"
            Interface "73e24986d7514_l"
        Port "97dbf2363c114_l"
            Interface "97dbf2363c114_l"
        Port "169681d9c9ef4_l"
            Interface "169681d9c9ef4_l"
    ovs_version: "2.13.8"
root@espoir:/home/espoir#
```

Affichant la liste des **ports**

root@espoir:/home/espoir# ovs-vsctl list-ifaces ovs-br0

```
root@espoir:/home/espoir# ovs-vsctl list-ifaces ovs-br0
169681d9c9ef4_l
73e24986d7514_l
97dbf2363c114_l
veth0
root@espoir:/home/espoir#
```

```
root@espoir:/home/espoir# ovs-vsctl list-ifaces ovs-br0
169681d9c9ef4_l
73e24986d7514_l
97dbf2363c114_l
veth0
root@espoir:/home/espoir#
```

Affichage de nom d'une l'interface par la commande suivante :

ovs-vsctl iface-to-br 169681d9c9ef4_l

```
root@espoir:/home/espoir# ovs-vsctl iface-to-br 169681d9c9ef4_l
ovs-br0
root@espoir:/home/espoir#
```

Test de connectivité entre les conteneurs docker 1 et docker 2 ensuite docker 3

Docker 1

```
root@7595d6fed04e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether d6:e1:3f:30:69:0a  txqueuelen 1000  (Ethernet)
        RX packets 39  bytes 5508 (5.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Docker 2

```
root@b52a8f59af6e:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 2a:4d:5e:f0:73:f5  txqueuelen 1000  (Ethernet)
        RX packets 40  bytes 5463 (5.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Docker 3

```
root@af20cea98808:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.13  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 66:25:50:0d:bc:68  txqueuelen 1000  (Ethernet)
        RX packets 36  bytes 5109 (5.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Connectivité entre docker 1 et docker 3

root@7595d6fed04e:/# ping -c3 192.168.1.13

```
root@7595d6fed04e:/# ping -c3 192.168.1.13
PING 192.168.1.13 (192.168.1.13) 56(84) bytes of data.
64 bytes from 192.168.1.13: icmp_seq=1 ttl=64 time=2.04 ms
64 bytes from 192.168.1.13: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 192.168.1.13: icmp_seq=3 ttl=64 time=0.163 ms

--- 192.168.1.13 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.112/0.770/2.037/0.895 ms
root@7595d6fed04e:/#
```

root@af20cea98808:/# ping -c3 192.168.1.11

```
root@af20cea98808:/# ping -c3 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=1.62 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.144 ms
64 bytes from 192.168.1.11: icmp_seq=3 ttl=64 time=0.093 ms

--- 192.168.1.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.093/0.617/1.615/0.705 ms
root@af20cea98808:/#
```

Connectivité entre docker 2 et docker 3

root@b52a8f59af6e:/# ping -c3 192.168.1.13

```
root@b52a8f59af6e:/# ping -c3 192.168.1.13
PING 192.168.1.13 (192.168.1.13) 56(84) bytes of data.
64 bytes from 192.168.1.13: icmp_seq=1 ttl=64 time=1.42 ms
64 bytes from 192.168.1.13: icmp_seq=2 ttl=64 time=0.085 ms
64 bytes from 192.168.1.13: icmp_seq=3 ttl=64 time=0.098 ms

--- 192.168.1.13 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2039ms
rtt min/avg/max/mdev = 0.085/0.535/1.423/0.627 ms
root@b52a8f59af6e:/#
```

root@af20cea98808:/# ping -c3 192.168.1.12

```
root@af20cea98808:/# ping -c3 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.
64 bytes from 192.168.1.12: icmp_seq=1 ttl=64 time=0.870 ms
64 bytes from 192.168.1.12: icmp_seq=2 ttl=64 time=0.149 ms
64 bytes from 192.168.1.12: icmp_seq=3 ttl=64 time=0.104 ms

--- 192.168.1.12 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.104/0.374/0.870/0.350 ms
root@af20cea98808:/#
```

2.4 Relier deux switch openvswitch

2.4.1 Étapes pour Connecter Deux Bridges Open vSwitch

2.4.1.1 Création des Patch Ports sur chaque Bridge

Pour ce faire, chaque **bridge** doit avoir un port **patch** qui sera connecté à l'autre bridge.

Et dans notre cas nous disposant de deux bridges (**openvswitch**) nommés **switch-noir** et **ovs-br0**

```
root@espoir:/home/espoir# ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge switch-noir
        Controller "tcp:127.0.0.1:6633"
        Port ens33
            Interface ens33
        Port switch-noir
            Interface switch-noir
                type: internal
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port ovs-br0
            Interface ovs-br0
                type: internal
        Port "73e24986d7514_l"
            Interface "73e24986d7514_l"
        Port "97dbf2363c114_l"
            Interface "97dbf2363c114_l"
        Port "169681d9c9ef4_l"
            Interface "169681d9c9ef4_l"
    ovs_version: "2.13.8"
```

root@espoir:/home/espoir# sudo ovs-vsctl add-port switch-noir patch-switch-noir-to-ovs-br0 -- set interface patch-switch-noir-to-ovs-br0 type=patch options:peer=patch-ovs-br0-to-switch-noir
root@espoir:/home/espoir# sudo ovs-vsctl add-port ovs-br0 patch-ovs-br0-to-switch-noir -- set interface patch-ovs-br0-to-switch-noir type=patch options:peer=patch-switch-noir-to-ovs-br0

```
root@espoir:/home/espoir# sudo ovs-vsctl add-port switch-noir patch-switch-noir-to-ovs-br0 -- set interface patch-switch-n
oir-to-ovs-br0 type=patch options:peer=patch-ovs-br0-to-switch-noir
root@espoir:/home/espoir# sudo ovs-vsctl add-port ovs-br0 patch-ovs-br0-to-switch-noir -- set interface patch-ovs-br0-to-s
witch-noir type=patch options:peer=patch-switch-noir-to-ovs-br0
root@espoir:/home/espoir#
```

2.4.1.2 Vérification

Après avoir exécuté les commandes ci-haut, vérifions que les deux bridges sont bien connectés

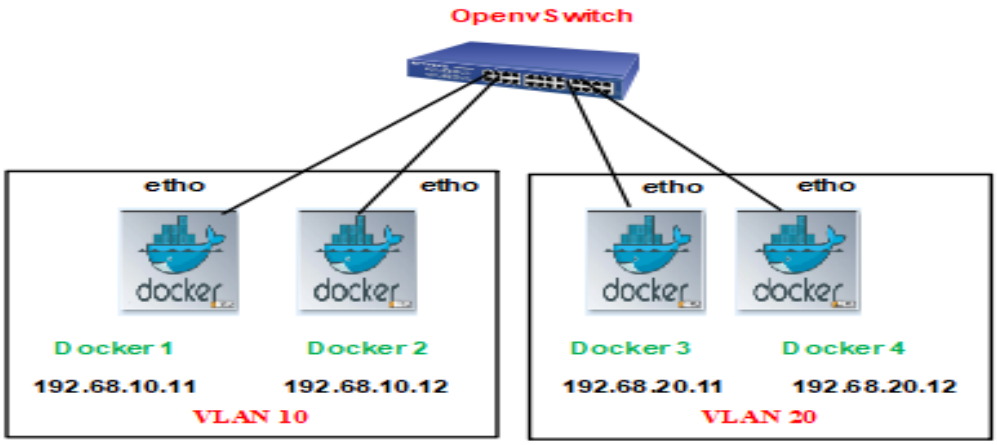
root@espoir:/home/espoir# sudo ovs-vsctl show

```
root@espoir:/home/espoir# sudo ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge switch-noir
        Controller "tcp:127.0.0.1:6633"
        Port patch-switch-noir-to-ovs-br0
            Interface patch-switch-noir-to-ovs-br0
                type: patch
                options: {peer=patch-ovs-br0-to-switch-noir}
        Port ens33
            Interface ens33
        Port switch-noir
            Interface switch-noir
                type: internal
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port patch-ovs-br0-to-switch-noir
            Interface patch-ovs-br0-to-switch-noir
                type: patch
                options: {peer=patch-switch-noir-to-ovs-br0}
        Port ovs-br0
            Interface ovs-br0
                type: internal
        Port "73e24986d7514_l"
            Interface "73e24986d7514_l"
        Port "97dbf2363c114_l"
            Interface "97dbf2363c114_l"
        Port "169681d9c9ef4_l"
            Interface "169681d9c9ef4_l"
    ovs_version: "2.13.8"
```

TP 3: ROUTAGE INTER-VLAN

L'objectif de ce TP est de :

1. Comprendre les concepts VLAN et routage inter-VLAN
2. Configurer les VLANs sur un Switch Openvsitch
3. Faire un Routage Inter-VLAN avec un switch Openvswitch



3.1 Création de switch openvswitch

On crée le switch noir par la commande

root@espoir:/home/espoir# ovs-vsctl add-br switch-noir

```
root@espoir:/home/espoir# ovs-vsctl add-br switch-noir
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@4c8b652be958:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@73d2274c6190:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1
root@c7af0ecc734d:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1
root@ec544c994439:/#
```

root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker1 --
ipaddress=192.168.10.11/24 --gateway=192.168.10.1
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker2 --
ipaddress=192.168.10.12/24 --gateway=192.168.10.1
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker3 --
ipaddress=192.168.20.11/24 --gateway=192.168.20.1
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker4 --
ipaddress=192.168.20.12/24 --gateway=192.168.20.1

```
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker1 --ipaddress=192.168.10.11/24 --gateway=192.168.10.1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker2 --ipaddress=192.168.10.12/24 --gateway=192.168.10.1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker3 --ipaddress=192.168.20.11/24 --gateway=192.168.20.1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker add-port switch-noir eth0 docker4 --ipaddress=192.168.20.12/24 --gateway=192.168.20.1
root@espoir:/home/espoir#
```

3.2 Affectation des machines dans les VLANs

On affecte les machines docker1 et docker2 dans le VLAN 10, et docker3 et docker4 dans le VLAN 20 avec les commandes suivantes :

VLAN 10

root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker1 10

root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker2 10

```
root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker1 10
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker2 10
root@espoir:/home/espoir#
```

- **VLAN 20**

root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker3 20

root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker4 20

```
root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker3 20
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker set-vlan switch-noir eth0 docker4 20
root@espoir:/home/espoir#
```

3.3 Configurer le port switch-noir en **mode trunk**

Pour configurer **switch-noir** en **mode trunk** et transporter les **VLANs 10** et **20**, exécutez la commande suivante :

root@espoir:/home/espoir# ovs-vsctl set port switch-noir trunks=10,20

```
root@espoir:/home/espoir# ovs-vsctl set port switch-noir trunks=10,20
root@espoir:/home/espoir#
```

On affiche les VLANs par la commande suivante :

root@espoir:/home/espoir# ovs-vsctl list port switch-noir

```
root@espoir:/home/espoir# ovs-vsctl list port switch-noir
_uuid               : 9b045f79-1bb2-48cd-9f0c-01f131aec180
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [8b0b1ed1-b8f1-4fbc-a5b1-3802f7fe9990]
lacp                : []
mac                 : []
name                : switch-noir
other_config        : {}
protected           : false
qos                 : []
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : [10, 20]
vlan_mode           : []
root@espoir:/home/espoir#
```

3.4 Redémarrage des conteneurs

Parfois, les modifications apportées aux interfaces réseau des conteneurs nécessitent un redémarrage des conteneurs pour prendre effet.

root@espoir:/home/espoir# docker restart docker1 docker2 docker3 docker4

```
root@espoir:/home/espoir# docker restart docker1 docker2 docker3 docker4
docker1
docker2
docker3
docker4
root@espoir:/home/espoir#
```

3.5 Teste entre les dockers de mêmes VLANs

Docker1 et docker2

Sur docker1, effectuons le ping sur docker2(192.168.10.12)

```
root@cc7aa20a4e87:/# ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=64 time=0.075 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=64 time=0.095 ms
^C
--- 192.168.10.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.075/0.085/0.095/0.010 ms
root@cc7aa20a4e87:/#
```

Doker3 et docker4

Sur docker3, on fait le ping sur docker4(192.168.20.12)

```
root@04fe8a37a72b:/# ping 192.168.20.12
PING 192.168.20.12 (192.168.20.12) 56(84) bytes of data.
64 bytes from 192.168.20.12: icmp_seq=1 ttl=64 time=0.743 ms
64 bytes from 192.168.20.12: icmp_seq=2 ttl=64 time=0.046 ms
^C
--- 192.168.20.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1011ms
rtt min/avg/max/mdev = 0.046/0.394/0.743/0.348 ms
root@04fe8a37a72b:/#
```

3.6 Transformation de notre machine en routeur

root@espoir:/home/espoir# ovs-vsctl add-port switch-noir vlan10 tag=10 -- set interface vlan10 type=internal

root@espoir:/home/espoir# ovs-vsctl add-port switch-noir vlan20 tag=20 -- set interface vlan20 type=internal

```
root@espoir:/home/espoir# ovs-vsctl add-port switch-noir vlan10 tag=10 -- set interface vlan10 type=internal
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-vsctl add-port switch-noir vlan20 tag=20 -- set interface vlan20 type=internal
root@espoir:/home/espoir#
```

On affecte des adresses IP aux interfaces virtuelles de VLAN

root@espoir:/home/espoir# ifconfig vlan10 192.168.10.1 up

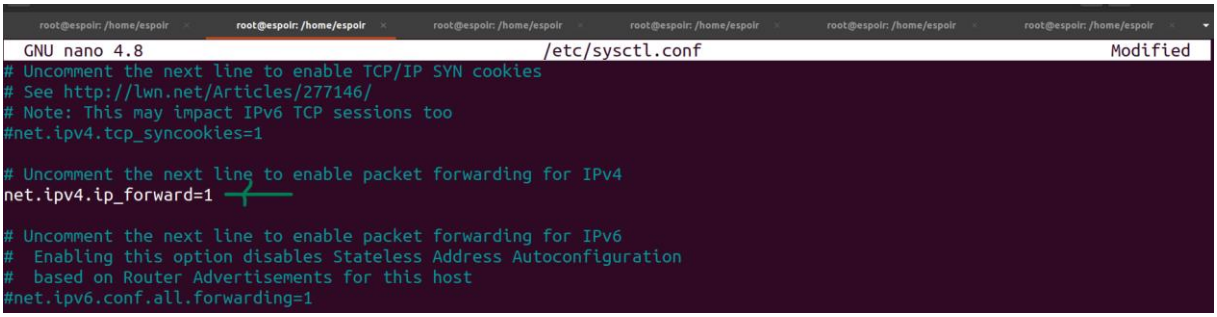
root@espoir:/home/espoir# ifconfig vlan20 192.168.20.1 up

```
root@espoir:/home/espoir# ifconfig vlan10 192.168.10.1 up
root@espoir:/home/espoir#
root@espoir:/home/espoir# ifconfig vlan20 192.168.20.1 up
root@espoir:/home/espoir#
```

On autorise les paquets à quitter une interface de notre machine pour aller vers une autre interface dans le fichier **/etc/sysctl.conf**

root@espoir:/home/espoir# nano /etc/sysctl.conf

```
root@espoir:/home/espoir# nano /etc/sysctl.conf
root@espoir:/home/espoir#
```



root@espoir:/home/espoir# sysctl -p

```
root@espoir:/home/espoir# sysctl -p
net.ipv4.ip_forward = 1
root@espoir:/home/espoir#
```

On transforme machine en passerelle

root@espoir:/home/espoir# iptables -A FORWARD -j ACCEPT

```
root@espoir:/home/espoir# iptables -A FORWARD -j ACCEPT
root@espoir:/home/espoir#
```

3.7 Verification des adresse IP attribuées

Docker1

root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1

root@cc7aa20a4e87:/# ifconfig

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@cc7aa20a4e87:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether ba:66:9b:c2:6b:f7  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4755 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Docker2

root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1

root@02105c1e3707:/# ifconfig

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@02105c1e3707:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether ea:6c:df:99:5e:8e  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4755 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Docker3

root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1

root@04fe8a37a72b:/# ifconfig

```
root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1
root@04fe8a37a72b:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether aa:c5:8b:ee:03:66  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4870 (4.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Docker4

root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1

root@d0d69f7b0815:/# ifconfig

```
root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1
root@d0d69f7b0815:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 9e:44:35:f6:20:bf  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 5002 (5.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- **Teste de routage inter vlan**

Sur docker 1, on effectue des pings sur docker 3(192.168.20.11) et docker4(192.168.20.12)

```
root@cc7aa20a4e87:/# ping 192.168.20.11
PING 192.168.20.11 (192.168.20.11) 56(84) bytes of data.
64 bytes from 192.168.20.11: icmp_seq=1 ttl=63 time=0.933 ms
64 bytes from 192.168.20.11: icmp_seq=2 ttl=63 time=0.097 ms
^C
--- 192.168.20.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.097/0.515/0.933/0.418 ms
root@cc7aa20a4e87:/# ping 192.168.20.12
PING 192.168.20.12 (192.168.20.12) 56(84) bytes of data.
64 bytes from 192.168.20.12: icmp_seq=1 ttl=63 time=0.697 ms
64 bytes from 192.168.20.12: icmp_seq=2 ttl=63 time=0.102 ms
^C
--- 192.168.20.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1027ms
rtt min/avg/max/mdev = 0.102/0.399/0.697/0.297 ms
root@cc7aa20a4e87:/#
```

Sur docker4, effectuons des ping sur docker1(192.168.10.11) et docker2(192.168.10.12)

```
root@d0d69f7b0815:/# ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=63 time=0.481 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=63 time=0.073 ms
^C
--- 192.168.10.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1018ms
rtt min/avg/max/mdev = 0.073/0.277/0.481/0.204 ms
root@d0d69f7b0815:/# ping 192.168.10.12
PING 192.168.10.12 (192.168.10.12) 56(84) bytes of data.
64 bytes from 192.168.10.12: icmp_seq=1 ttl=63 time=0.664 ms
64 bytes from 192.168.10.12: icmp_seq=2 ttl=63 time=0.117 ms
^C
--- 192.168.10.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.117/0.390/0.664/0.273 ms
root@d0d69f7b0815:/#
```

Sur docker3, faisant des pings sur docke1(192.168.10.11) et docker2(192.168.10.12)

```
root@04fe8a37a72b:/# ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=63 time=0.564 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=63 time=0.060 ms
^C
--- 192.168.10.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.060/0.312/0.564/0.252 ms
root@04fe8a37a72b:/# ping 192.168.10.12
PING 192.168.10.12 (192.168.10.12) 56(84) bytes of data.
64 bytes from 192.168.10.12: icmp_seq=1 ttl=63 time=0.649 ms
64 bytes from 192.168.10.12: icmp_seq=2 ttl=63 time=0.057 ms
^C
--- 192.168.10.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1042ms
rtt min/avg/max/mdev = 0.057/0.353/0.649/0.296 ms
root@04fe8a37a72b:/#
```

Et enfin, sur docker2 on fait des pings sur docker3(192.168.20.11) et docker4(192.168.20.12)

```
root@02105c1e3707:/# ping 192.168.20.11
PING 192.168.20.11 (192.168.20.11) 56(84) bytes of data.
64 bytes from 192.168.20.11: icmp_seq=1 ttl=63 time=0.652 ms
64 bytes from 192.168.20.11: icmp_seq=2 ttl=63 time=0.112 ms
^C
--- 192.168.20.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1033ms
rtt min/avg/max/mdev = 0.112/0.382/0.652/0.270 ms
root@02105c1e3707:/# ping 192.168.20.12
PING 192.168.20.12 (192.168.20.12) 56(84) bytes of data.
64 bytes from 192.168.20.12: icmp_seq=1 ttl=63 time=0.494 ms
64 bytes from 192.168.20.12: icmp_seq=2 ttl=63 time=0.067 ms
^C
--- 192.168.20.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1022ms
rtt min/avg/max/mdev = 0.067/0.280/0.494/0.213 ms
root@02105c1e3707:/#
```

root@espoir:/home/espoir# ovs-vsctl show

```
root@espoir:/home/espoir# ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge switch-noir
        Port "061600815c794_l"
            tag: 10
            Interface "061600815c794_l"
        Port "24873151cd3b4_l"
            tag: 20
            Interface "24873151cd3b4_l"
        Port "753ffbe5d8814_l"
            tag: 20
            Interface "753ffbe5d8814_l"
        Port vlan10
            tag: 10
            Interface vlan10
                type: internal
        Port switch-noir
            trunks: [10, 20]
            Interface switch-noir
                type: internal
        Port f9b09f7fd88c4_l
            tag: 10
            Interface f9b09f7fd88c4_l
        Port vlan20
            tag: 20
            Interface vlan20
                type: internal
    ovs_version: "2.13.8"
root@espoir:/home/espoir#
```

TP 4: CASCADER DES SWITCHES OPENVSWITCH

Ce TP a pour but de se familiariser avec la configuration et la gestion de switches virtuels OpenvSwitch (OVS) dans un environnement réseau.

On crée les deux switches ovs-br0 et ovs-br1

root@espoir:/home/espoir# ovs-vsctl add-br ovs-br1

root@espoir:/home/espoir#

root@espoir:/home/espoir# ovs-vsctl add-br ovs-br0

root@espoir:/home/espoir#

```
root@espoir:/home/espoir# ovs-vsctl add-br ovs-br1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-vsctl add-br ovs-br0
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# ovs-vsctl show

```
root@espoir:/home/espoir# ovs-vsctl show
124f7c01-ddc0-4fa4-ad32-9ede69b528eb
    Bridge ovs-br0
        Port ovs-br0
            Interface ovs-br0
                type: internal
    Bridge ovs-br1
        Port ovs-br1
            Interface ovs-br1
                type: internal
    ovs_version: "2.13.8"
root@espoir:/home/espoir#
```

On ajoute les machines puis on les donne les adresses IP :

root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@c6999cfc9409:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@5659823b1bd5:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1
root@443f4a740132:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1
root@b18e3107beef:/#
```

Les machines docker1 et docker2 sont dans le switch **ovs-br0**

root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker1 --

ipaddress=192.168.1.11/24 --gateway=192.168.1.1

root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker2 --

ipaddress=192.168.1.12/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker1 --ipaddress=192.168.1.11/24 --gateway=192.168.1.1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker add-port ovs-br0 eth0 docker2 --ipaddress=192.168.1.12/24 --gateway=192.168.1.1
root@espoir:/home/espoir#
```

Les machines docker3 et docker4 sont dans le switch **ovs-br1**

root@espoir:/home/espoir# ovs-docker add-port ovs-br1 eth0 docker3 --

ipaddress=192.168.1.13/24 --gateway=192.168.1.1

root@espoir:/home/espoir# ovs-docker add-port ovs-br1 eth0 docker4 --

ipaddress=192.168.1.14/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# ovs-docker add-port ovs-br1 eth0 docker3 --ipaddress=192.168.1.13/24 --gateway=192.168.1.1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-docker add-port ovs-br1 eth0 docker4 --ipaddress=192.168.1.14/24 --gateway=192.168.1.1
root@espoir:/home/espoir#
```

4.1 Création d'une extrémité du câble virtuel sur OVS-BR0 du nom de patch2br0

root@espoir:/home/espoir# ovs-vsctl add-port ovs-br0 patch2br0 -- set interface patch2br0 type=patch options:peer=patch2br1

root@espoir:/home/espoir# ovs-vsctl add-port ovs-br1 patch2br1 -- set interface patch2br1 type=patch options:peer=patch2br0

```
root@espoir:/home/espoir# ovs-vsctl add-port ovs-br0 patch2br0 -- set interface patch2br0 type=patch options:peer=patch2br1
root@espoir:/home/espoir#
root@espoir:/home/espoir# ovs-vsctl add-port ovs-br1 patch2br1 -- set interface patch2br1 type=patch options:peer=patch2br0
root@espoir:/home/espoir#
```

On se place sur docker4 connecté au switch ovs-br1 pour pinguer docker1 connecté au switch ovs-br0 :

```
root@espoir:/home/espoir# docker run -it --net=none --name docker4 ubuntu1
root@b18e3107beef:/#
root@b18e3107beef:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.14  netmask 255.255.255.0  broadcast 0.0.0.0
        ether fe:95:62:10:d0:49  txqueuelen 1000  (Ethernet)
        RX packets 30  bytes 4578 (4.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@b18e3107beef:/# ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=17.3 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.067 ms
^C
--- 192.168.1.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.067/8.678/17.289/8.611 ms
root@b18e3107beef:/#
```

Ensuite, sur docker1 connecté au switch ovs-br0 pour faire le ping docker4 connecté au switch ovs-br1 :

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@c6999cfc9409:/#
root@c6999cfc9409:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether e2:43:12:86:ec:0c  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 4755 (4.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@c6999cfc9409:/# ping 192.168.1.14
PING 192.168.1.14 (192.168.1.14) 56(84) bytes of data.
64 bytes from 192.168.1.14: icmp_seq=1 ttl=64 time=0.317 ms
64 bytes from 192.168.1.14: icmp_seq=2 ttl=64 time=0.050 ms
^C
--- 192.168.1.14 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.050/0.183/0.317/0.133 ms
root@c6999cfc9409:/#
```

TP 5: PORT MIRRORING ET FOREWARDING

Pour réaliser le **TP 6** en utilisant **Open vSwitch** (**OVS)** avec le port mirroring et le **forwarding** avec le bridge **switch-noir**, voici un guide détaillé en utilisant vos données :

5.1 **Configuration des Conteneurs Docker**

Tout d'abord, on doit avoir des conteneurs **Docker** que nous les avons lancées avec le réseau **none**. On les ajoute à un bridge **OVS** :

root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker1 ubuntu1
root@28dff29fe22e:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker2 ubuntu1
root@2a5e9d1382fa:/#
```

root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1

```
root@espoir:/home/espoir# docker run -it --net=none --name docker3 ubuntu1
root@9687caf25272:/#
```

5.2 Création d'un bridge OVS nommé switch-noir

root@espoir:/home/espoir# sudo ovs-vsctl add-br switch-noir

```
root@espoir:/home/espoir# sudo ovs-vsctl add-br switch-noir
root@espoir:/home/espoir#
```

5.3 Ajout des ports Docker au bridge OVS

On configure les conteneurs avec des adresses **IP** et une passerelle

root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker1 --ipaddress=192.168.1.11/24 --gateway=192.168.1.1

root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker2 --ipaddress=192.168.1.12/24 --gateway=192.168.1.1

root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker3 –ipaddress=192.168.1.13/24 --gateway=192.168.1.1

```
root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker1 --ipaddress=192.168.1.11/24 --ga
teway=192.168.1.1
root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker2 --ipaddress=192.168.1.12/24 --ga
teway=192.168.1.1
root@espoir:/home/espoir# sudo ovs-docker add-port switch-noir eth0 docker3 --ipaddress=192.168.1.13/24 --ga
teway=192.168.1.1
root@espoir:/home/espoir#
```

root@espoir:/home/espoir# sudo ovs-vsctl add-port switch-noir veth0 -- set interface veth0 type=internal

root@espoir:/home/espoir# ip address add 192.168.1.1/24 dev veth0

```
root@espoir:/home/espoir# sudo ovs-vsctl add-port switch-noir veth0 -- set interface veth0 type=internal
root@espoir:/home/espoir#
root@espoir:/home/espoir# ip address add 192.168.1.1/24 dev veth0
root@espoir:/home/espoir#
```

5.4 Création et Configuration du Miroir

5.4.1 Créez le miroir dans OVS

root@espoir:/home/espoir# sudo ovs-vsctl -- --id=@m create mirror name=ec2ltmir -- add bridge switch-noir mirrors @m

```
root@espoir:/home/espoir# sudo ovs-vsctl -- --id=@m create mirror name=ec2ltmir -- add bridge switch-noir mirrors @m
a3703578-5709-479d-9f32-f7ff8657aec0
root@espoir:/home/espoir#
```

5.4.2 Listez les ports disponibles

On liste les ports afin d'obtenir leurs identifiants car on aura besoin des identifiants des ports connectés au bridge :

```
root@espoir:/home/espoir# sudo ovs-vsctl list port
_uuid               : ff196b79-f42d-4c98-955c-98c663761075
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [e91648a3-7da6-4d4c-aa76-2839ae5b2784]
lacp                : []
mac                 : []
name                : "75b92e8e6aca4_l"
```

```
_uuid               : 5929c512-3fbc-466c-abee-bfe2e830dc9e
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [7b9dfb97-a792-4493-a260-cf9da8bf3f29]
lacp                : []
mac                 : []
name                : "587915b455b94_l"
```

```
_uuid               : 48a7eccf-cc57-4f74-84a7-b214249a6dab
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [aa5f9e83-df98-4d53-bba4-5adf02d56445]
lacp                : []
mac                 : []
name                : d266b1284a1e4_l
```

5.4.3 Identifier les conteneurs correspondants aux interfaces

On peut vérifier quel conteneur correspond à ces identifiants en listant les conteneurs Docker avec leurs IDs

root@espoir:/home/espoir# sudo docker ps -a --no-trunc

```
root@espoir:/home/espoir# sudo docker ps -a --no-trunc
CONTAINER ID                                                       IMAGE
     COMMAND       CREATED          STATUS                     PORTS     NAMES
864849e017581ecfce5256b4be76357bb6bb1425c19e419464d79974f02ba950   ubuntu1
     "/bin/bash"   12 minutes ago   Up 12 minutes                        docker3
0921b7df0b2101e4bc64cc9c179a21f20f4b40a9c82729b8a5e613a87aa44936   ubuntu1
     "/bin/bash"   12 minutes ago   Up 12 minutes                        docker2
7f8e4057dc8189f021cb66c29a80f30ad5049b742bda4e50a73ef18a946ae700   ubuntu1
     "/bin/bash"   13 minutes ago   Up 13 minutes                        docker1
```

5.4.4 Obtenez les ports à utiliser pour le mirroring

Nous devons sélectionner les ports source et destination qu'on souhaite surveiller. Si nous voulons surveiller tout le trafic sur certains ports, dans notre cas nous allons surveiller le trafic entre **docker1** (**192.168.1.11)** et **docker2** (**192.168.1.12**) :

- Pour docker1 : Port bd112e04c70b4_l (ID : d0e74340-cf03-4133-bf79-2d8b192debaa)
- Pour docker2 : Port 0e4ec491f1de4_l (ID : fe8581d9-4634-4172-ac4d-d6b387bb1e6c)

root@espoir:/home/espoir# sudo ovs-vsctl show

```
root@espoir:/home/espoir# sudo ovs-vsctl show
2bf809fa-7c35-41ae-a172-93fb37b09407
    Bridge switch-noir
        Port switch-noir
            Interface switch-noir
                type: internal
        Port d266b1284a1e4_l
            Interface d266b1284a1e4_l  1
        Port "587915b455b94_l"
            Interface "587915b455b94_l" 2
        Port "75b92e8e6aca4_l"
            Interface "75b92e8e6aca4_l" 3
    ovs_version: "2.13.8"
```

root@espoir:/home/espoir# sudo ovs-vsctl list port d266b1284a1e4_l

```
root@espoir:/home/espoir# sudo ovs-vsctl list port d266b1284a1e4_l
_uuid               : 48a7eccf-cc57-4f74-84a7-b214249a6dab
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [aa5f9e83-df98-4d53-bba4-5adf02d56445]
lacp                : []
mac                 : []
name                : d266b1284a1e4_l
other_config        : {}
protected           : false
qos                 : []
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
root@espoir:/home/espoir#
```

_uuid : 48a7eccf-cc57-4f74-84a7-b214249a6dab

root@espoir:/home/espoir# sudo ovs-vsctl list port 587915b455b94_l

```
root@espoir:/home/espoir# sudo ovs-vsctl list port 587915b455b94_l
_uuid               : 5929c512-3fbc-466c-abee-bfe2e830dc9e
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [7b9dfb97-a792-4493-a260-cf9da8bf3f29]
lacp                : []
mac                 : []
name                : "587915b455b94_l"
other_config        : {}
protected           : false
qos                 : []
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
```

_uuid : 5929c512-3fbc-466c-abee-bfe2e830dc9e

Notez les IDs des ports correspondant aux interfaces des conteneurs Docker.

5.4.5 Configurez le miroir avec les ports source et destination

On configure le port de sortie du miroir en remplaçant **<output_port_id>** dans la commande suivante : **sudo ovs-vsctl set mirror ec2ltmir output_port=<output_port_id>** par l'**ID** du port où l'on souhaite envoyer le trafic miroir. Dans notre cas c'est le port : 75b92e8e6aca4_l et le _uuid : ff196b79-f42d-4c98-955c-98c663761075

root@espoir:/home/espoir# sudo ovs-vsctl list port 75b92e8e6aca4_l

```
root@espoir:/home/espoir# sudo ovs-vsctl list port 75b92e8e6aca4_l
_uuid               : ff196b79-f42d-4c98-955c-98c663761075
bond_active_slave   : []
bond_downdelay      : 0
bond_fake_iface     : false
bond_mode           : []
bond_updelay        : 0
cvlans              : []
external_ids        : {}
fake_bridge         : false
interfaces          : [e91648a3-7da6-4d4c-aa76-2839ae5b2784]
lacp                : []
mac                 : []
name                : "75b92e8e6aca4_l"
other_config        : {}
protected           : false
qos                 : []
rstp_statistics     : {}
rstp_status         : {}
statistics          : {}
status              : {}
tag                 : []
trunks              : []
vlan_mode           : []
```

_uuid : ff196b79-f42d-4c98-955c-98c663761075

root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir output_port= ff196b79-f42d-4c98-955c-98c663761075

```
root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir output_port=ff196b79-f42d-4c98-955c-98c663761075
root@espoir:/home/espoir#
root@espoir:/home/espoir#
```

Ensuite, on configure le port source

root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir select_src_port=48a7eccf-cc57-4f74-84a7-b214249a6dab

```
root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir select_src_port=48a7eccf-cc57-4f74-84a7-b214249a6dab
root@espoir:/home/espoir#
```

Et on configure le port destination

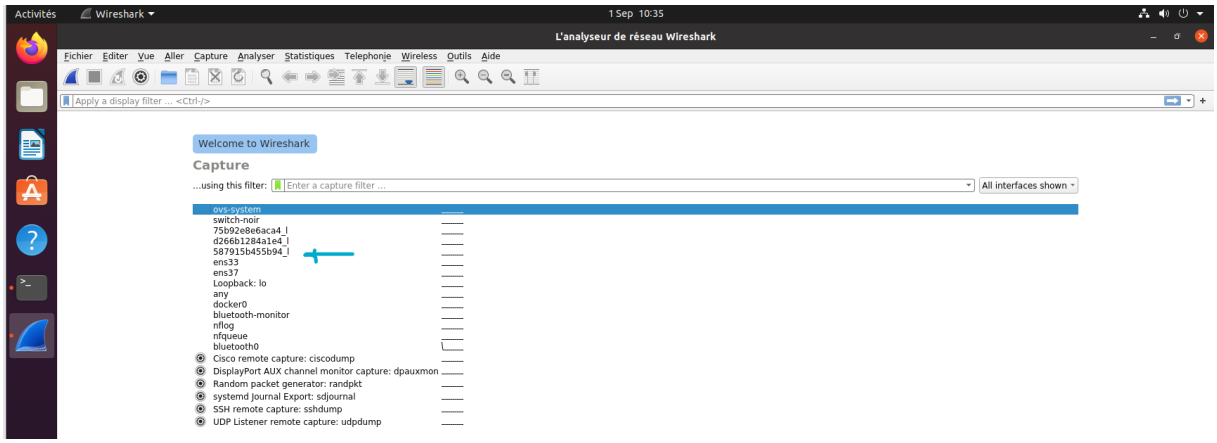
root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir select_dst_port=5929c512-3fbc-466c-abee-bfe2e830dc9e

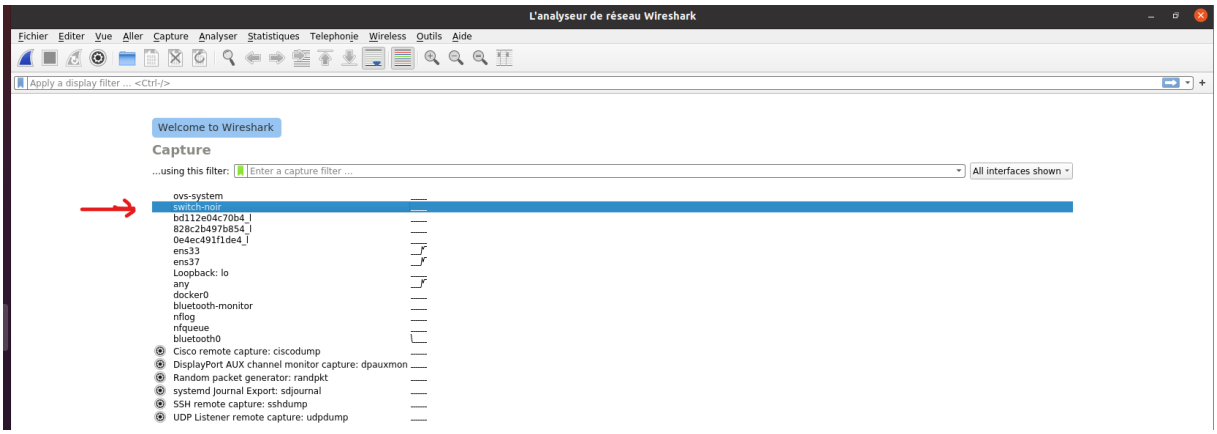
```
root@espoir:/home/espoir# sudo ovs-vsctl set mirror ec2ltmir select_dst_port=5929c512-3fbc-466c-abee-bfe2e830dc9e
root@espoir:/home/espoir#
```

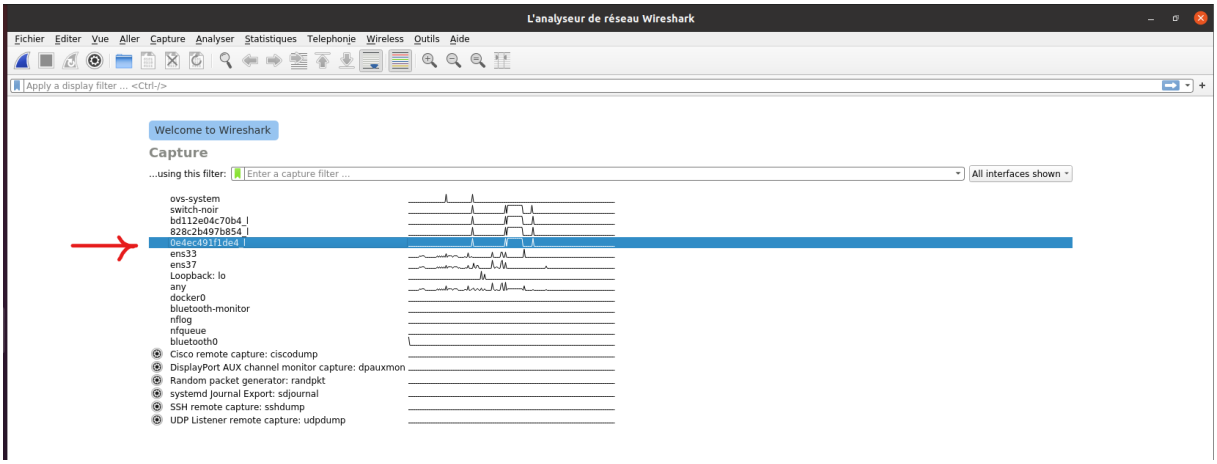
Cette commande permet de retirer les miroirs des ponts **sudo ovs-vsctl -- clear Bridge switch-noir mirrors**

5.4.6 Installation de wireshark et tcpdump

```
root@espoir:/home/espoir# apt install wireshark
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
wireshark est déjà la version la plus récente (3.2.3-1).
0 mis à jour, 0 nouvellement installés, 0 à enlever et 23 non mis à jour.
root@espoir:/home/espoir#
```







5.4.7 Test de la connectivité

On accède aux conteneurs **Docker** et testez la connectivité entre eux **:**

Depuis docker1 on fait un **ping** sur **docker2**

Depuis docker2 on fait aussi un **ping** sur le **docker1**

Partie II: VxLAN

II.1 Introduction à VXLAN (Virtual Extensible LAN)

Le **VXLAN (Virtual Extensible LAN)** est une technologie de virtualisation de réseau qui étend les réseaux locaux (LAN) en créant des réseaux virtuels superposés sur une infrastructure réseau physique. Il s'agit d'un protocole de tunneling de niveau 2 conçu pour répondre aux limitations des réseaux traditionnels VLAN (Virtual Local Area Network) en offrant une plus grande évolutivité et en permettant la création de réseaux virtuels étendus au-delà des frontières physiques des centres de données.

Dans un contexte où les **datacenters** modernes nécessitent un nombre croissant de machines virtuelles et de réseaux isolés, VXLAN permet de répondre aux défis posés par la limite des **4096 VLANs** disponibles avec le standard **IEEE 802.1Q**. Avec VXLAN, il est possible de créer jusqu'à **16 millions** de réseaux logiques (ou identifiants **VXLAN - VNI**), ce qui le rend beaucoup plus adapté aux environnements cloud ou multi-tenant.

II.2 Fonctionnement de VXLAN

VXLAN encapsule les trames Ethernet dans des paquets UDP (**User Datagram Protocol**) qui peuvent être acheminés à travers un réseau IP existant. Cela permet à un réseau de niveau 2 (**L2**) d'être transporté sur un réseau de niveau 3 (**L3**), ce qui signifie que les VLANs peuvent être étendus à travers des réseaux IP traditionnels.

Voici les composants clés de VXLAN :

- **VXLAN Network Identifier (VNI)** : il s'agit de l'identifiant unique attribué à chaque réseau virtuel. Un VNI est un champ de 24 bits, permettant de créer jusqu'à 16 millions de réseaux logiques distincts.
- **VXLAN Tunnel Endpoint (VTEP)** : les VTEPs sont des points d'extrémité de tunnel VXLAN. Ils sont responsables de l'encapsulation et de la décapsulation des trames Ethernet dans les paquets VXLAN. Les VTEPs se situent généralement au niveau des hyperviseurs ou des commutateurs d'accès.
- **Encapsulation** : lorsqu'une trame Ethernet est envoyée entre deux hôtes appartenant au même VNI, elle est encapsulée dans un en-tête VXLAN, puis transmise sous forme de paquet UDP sur un réseau IP. Cela permet de transporter plusieurs réseaux virtuels isolés au-dessus du réseau physique sans interférer avec ce dernier.
- **Multicast ou Unicast** : VXLAN utilise initialement le multicast pour diffuser des informations de type broadcast ou multidiffusion sur le réseau. Cependant, des implémentations plus récentes utilisent des techniques de unicast pour optimiser la diffusion du trafic et minimiser les charges réseau inutiles.

II.3 Avantages de VXLAN

- **Évolutivité** : l'un des principaux avantages de VXLAN est son évolutivité. Alors que les VLANs sont limités à 4096 ID, VXLAN permet de créer jusqu'à 16 millions de réseaux logiques, ce qui est essentiel dans les environnements de datacenter et cloud.
- **Extensibilité géographique** : en encapsulant les trames Ethernet sur un réseau IP, VXLAN permet d'étendre des réseaux de niveau 2 au-delà des frontières physiques des datacenters, facilitant ainsi la mobilité des machines virtuelles entre plusieurs sites.
- **Isolation des locataires (multi-tenant)** : VXLAN est très utile dans les environnements multi-tenant où de nombreux clients ou applications partagent la même infrastructure

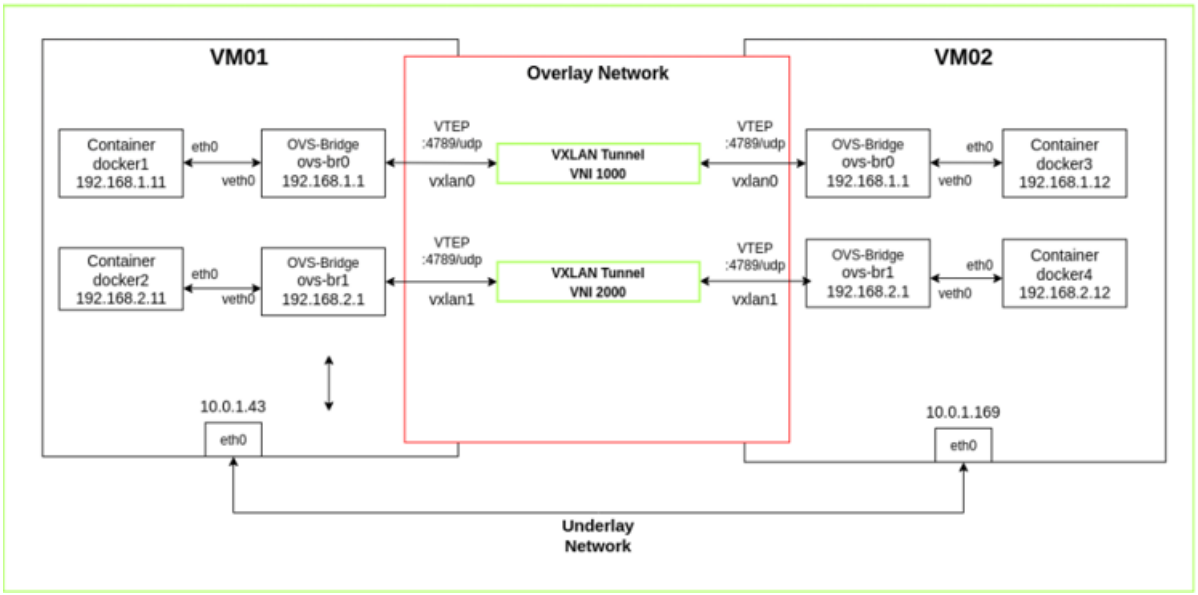
physique. Chaque réseau logique est isolé à l'aide d'un VNI distinct, ce qui garantit une séparation et une sécurité optimales entre les différents locataires.

- **Compatibilité avec les réseaux existants** : VXLAN fonctionne au-dessus des infrastructures IP standard, ce qui signifie qu'il peut être déployé sans modifications majeures des réseaux physiques existants. Cela en fait une solution flexible et adaptable.

II.4 Cas d'utilisation de VXLAN

- **Centres de données et cloud computing** : VXLAN est largement utilisé dans les datacenters pour l'extension des réseaux virtuels et pour la gestion des machines virtuelles dans des environnements cloud, offrant une isolation et une évolutivité élevées.
- **Mobilité des machines virtuelles (VM)** : VXLAN permet aux machines virtuelles de se déplacer librement entre différents serveurs physiques sans interrompre la connectivité réseau, en maintenant leurs adresses IP et MAC.
- **Environnements multi-tenant** : dans les environnements où plusieurs utilisateurs partagent la même infrastructure, VXLAN assure l'isolation complète des réseaux virtuels des différents locataires.

TP1: Mise en place de VxLANover VPN



Installation

On installe sur les 2 machines les outils **net-tools docker** et **openvswitch**

Machine 1

```
root@machine1:/home/elvis# apt update
Atteint :1 http://sn.archive.ubuntu.com/ubuntu focal InRelease
Atteint :2 http://sn.archive.ubuntu.com/ubuntu focal-updates InRelease
Atteint :3 http://sn.archive.ubuntu.com/ubuntu focal-backports InRelease
Atteint :4 http://security.ubuntu.com/ubuntu focal-security InRelease
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
14 paquets peuvent être mis à jour. Exécutez « apt list --upgradable » pour les voir.
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# sudo apt -y install net-tools docker.io openvswitch-switch

```
root@machine1:/home/elvis# sudo apt -y install net-tools docker.io openvswitch-switch
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
net-tools est déjà la version la plus récente (1.60+git20180626.aebd88e-1ubuntu1).
Les paquets supplémentaires suivants seront installés :
  bridge-utils containerd libunbound8 openvswitch-common pigz python3-openvswitch
  python3-sortedcontainers runc ubuntu-fan
Paquets suggérés :
```

Machine 2

```
root@machine2:/home/elvis# apt update
Atteint :1 http://sn.archive.ubuntu.com/ubuntu focal InRelease
Atteint :2 http://sn.archive.ubuntu.com/ubuntu focal-updates InRelease
Atteint :3 http://sn.archive.ubuntu.com/ubuntu focal-backports InRelease
Atteint :4 http://security.ubuntu.com/ubuntu focal-security InRelease
Atteint :5 https://dl.google.com/linux/chrome/deb stable InRelease
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Tous les paquets sont à jour.
```

root@machine2:/home/elvis# sudo apt -y install net-tools docker.io openvswitch-switch

```
root@machine2:/home/elvis# sudo apt -y install net-tools docker.io openvswitch-switch
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
net-tools est déjà la version la plus récente (1.60+git20180626.aebd88e-1ubuntu1).
openvswitch-switch est déjà la version la plus récente (2.13.8-0ubuntu1.4).
docker.io est déjà la version la plus récente (24.0.7-0ubuntu2~20.04.1).
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  galera-3 libconfig-inifiles-perl libdbd-mysql-perl libdbi-perl libmysqlclient21 libreadline5 libsnappy1v5 libterm-readkey-perl mariadb-common socat
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
root@machine2:/home/elvis#
```

- **Étape 1 : on crée les 2 switches sur chaque machine**

Machine 2

root@machine2:/home/elvis# ovs-vsctl add-br ovs-br0

root@machine2:/home/elvis# ovs-vsctl add-br ovs-br1

```
root@machine2:/home/elvis# ovs-vsctl add-br ovs-br0
root@machine2:/home/elvis# ovs-vsctl add-br ovs-br1
root@machine2:/home/elvis#
```

Machine 1

root@machine1:/home/elvis# ovs-vsctl add-br ovs-br1

root@machine1:/home/elvis# ovs-vsctl add-br ovs-br0

```
root@machine1:/home/elvis# ovs-vsctl add-br ovs-br1
root@machine1:/home/elvis# ovs-vsctl add-br ovs-br0
root@machine1:/home/elvis#
```

On crée les 2 interfaces internes sur les machines

root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal

root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br1 veth1 -- set interface veth1 type=internal

```
root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal
root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br1 veth1 -- set interface veth1 type=internal
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# sudo ovs-vsctl show

```
root@machine1:/home/elvis# sudo ovs-vsctl show
5abe198f-f94d-4cb6-bb8e-2048e8239ae0
    Bridge ovs-br1
        Port veth1
            Interface veth1
                type: internal
        Port ovs-br1
            Interface ovs-br1
                type: internal
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port ovs-br0
            Interface ovs-br0
                type: internal
    ovs_version: "2.13.8"
root@machine1:/home/elvis#
```

root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal

root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br1 veth1 -- set interface veth1 type=internal

```
root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br0 veth0 -- set interface veth0 type=internal
root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br1 veth1 -- set interface veth1 type=internal
root@machine2:/home/elvis#
```

root@machine2:/home/elvis# ovs-vsctl show

```
root@machine2:/home/elvis# ovs-vsctl show
5a92bd1c-7028-4f56-8a89-68ccd8ed954d
    Bridge ovs-br0
        Port veth0
            Interface veth0
                type: internal
        Port ovs-br0
            Interface ovs-br0
                type: internal
    Bridge ovs-br1
        Port veth1
            Interface veth1
                type: internal
        Port ovs-br1
            Interface ovs-br1
                type: internal
    ovs_version: "2.13.8"
root@machine2:/home/elvis#
```

On donne des adresses à ces interfaces et on les active sur les machines

root@machine2:/home/elvis# sudo ip address add 192.168.1.1/24 dev veth0

root@machine2:/home/elvis# ip address add 192.168.2.1/24 dev veth1

```
root@machine2:/home/elvis# sudo ip address add 192.168.1.1/24 dev veth0
root@machine2:/home/elvis# ip address add 192.168.2.1/24 dev veth1
root@machine2:/home/elvis#
```

- **Pour vérifier si l'adresse IP**

root@machine2:/home/elvis# ip addr show veth1

root@machine2:/home/elvis# ip addr show veth0

```
root@machine2:/home/elvis# ip addr show veth1
8: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 6a:6d:c1:64:da:d8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 scope global veth1
       valid_lft forever preferred_lft forever
root@machine2:/home/elvis#
root@machine2:/home/elvis# ip addr show veth0
7: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether c6:66:e0:52:3d:e6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 scope global veth0
       valid_lft forever preferred_lft forever
root@machine2:/home/elvis#
```

root@machine1:/home/elvis# sudo ip address add 192.168.1.1/24 dev veth0

root@machine1:/home/elvis# ip address add 192.168.2.1/24 dev veth1

```
root@machine1:/home/elvis# sudo ip address add 192.168.1.1/24 dev veth0
root@machine1:/home/elvis# ip address add 192.168.2.1/24 dev veth1
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# ip addr show veth1

root@machine1:/home/elvis# ip addr show veth0

```
root@machine1:/home/elvis# ip addr show veth1
8: veth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether ea:3c:59:bf:4c:40 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 scope global veth1
       valid_lft forever preferred_lft forever
root@machine1:/home/elvis#
root@machine1:/home/elvis# ip addr show veth0
7: veth0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 0a:0e:d4:33:5f:95 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 scope global veth0
       valid_lft forever preferred_lft forever
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# ip link set dev veth0 up mtu 1450

root@machine1:/home/elvis# ip link set dev veth1 up mtu 1450

```
root@machine1:/home/elvis# ip link set dev veth0 up mtu 1450
root@machine1:/home/elvis# ip link set dev veth1 up mtu 1450
root@machine1:/home/elvis#
```

Pour vérifier si l'interfaceveth0 a été mise en place (up) avec une MTU de 1450

root@machine1:/home/elvis# ip link show veth0

root@machine1:/home/elvis# ip link show veth1

```
root@machine1:/home/elvis# ip link show veth0
7: veth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 0a:0e:d4:33:5f:95 brd ff:ff:ff:ff:ff:ff
root@machine1:/home/elvis#
root@machine1:/home/elvis# ip link show veth1
8: veth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether ea:3c:59:bf:4c:40 brd ff:ff:ff:ff:ff:ff
root@machine1:/home/elvis#
```

root@machine2:/home/elvis# ip link set dev veth0 up mtu 1450

root@machine2:/home/elvis# ip link set dev veth1 up mtu 1450

```
root@machine2:/home/elvis# ip link set dev veth0 up mtu 1450
root@machine2:/home/elvis# ip link set dev veth1 up mtu 1450
root@machine2:/home/elvis#
```

- **Etape 2 : téléchargement des images debian et apporter des modifications pour intégrer des outils de test réseaux sur les 2 machines**

On télécharge l'image **debian**

root@machine1:/home/elvis# docker pull debian:latest

```
root@machine1:/home/elvis# docker pull debian:latest
latest: Pulling from library/debian
8cd46d290033: Pull complete
Digest: sha256:b8084b1a576c5504a031936e1132574f4ce1d6cc7130bbcc25a28f074539ae6b
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# docker images

```
root@machine1:/home/elvis# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
debian       latest    4fd3f4b75df3   4 days ago    117MB
root@machine1:/home/elvis#
```

root@machine2:/home/elvis# docker pull debian:latest

```
root@machine2:/home/elvis# docker pull debian:latest
latest: Pulling from library/debian
8cd46d290033: Pull complete
Digest: sha256:b8084b1a576c5504a031936e1132574f4ce1d6cc7130bbcc25a28f074539ae6b
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
root@machine2:/home/elvis#
```

Pour vérifier si l'image Debian est bien présente sur votre système, utilisez la commande :

root@machine2:/home/elvis# docker images

```
root@machine2:/home/elvis# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
debian       latest    4fd3f4b75df3   4 days ago    117MB
root@machine2:/home/elvis#
```

- **Redémarrez Docker :**

root@machine2:/home/elvis# sudo systemctl restart docker

```
root@machine2:/home/elvis# sudo systemctl restart docker
root@machine2:/home/elvis#
```

root@machine1:/home/elvis# sudo systemctl restart docker

```
root@machine1:/home/elvis# sudo systemctl restart docker
root@machine1:/home/elvis#
```

On lance un conteneur à partir duquel, on crée une nouvelle image **debian1**

root@machine1:/home/elvis# docker run -it debian:latest /bin/bash

```
root@machine1:/home/elvis# docker run -it debian:latest /bin/bash
root@e446841ba0fb:/#
```

root@machine2:/home/elvis# docker run -it debian:latest /bin/bash

```
root@machine2:/home/elvis# docker run -it debian:latest /bin/bash
root@b46a9326164d:/#
```

Sur le conteneur

```
root@machine2:/home/elvis# docker run -it debian:latest /bin/bash
root@b46a9326164d:/#
root@b46a9326164d:/# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8787 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.8 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [179 kB]
Fetched 9235 kB in 5s (1987 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@b46a9326164d:/#
```

```
root@machine1:/home/elvis# docker run -it debian:latest /bin/bash
root@e446841ba0fb:/#
root@e446841ba0fb:/# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 k
B]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8787 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.8 k
B]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packag
es [179 kB]
Fetched 9235 kB in 6s (1645 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@e446841ba0fb:/#
```

root@b46a9326164d:/# apt install net-tools iputils-ping

```
root@b46a9326164d:/# apt install net-tools iputils-ping
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcap2-bin libpam-cap
The following NEW packages will be installed:
  iputils-ping libcap2-bin libpam-cap net-tools
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 339 kB of archives.
After this operation, 1313 kB of additional disk space will be used.
```

root@e446841ba0fb:/# apt install net-tools iputils-ping

```
root@e446841ba0fb:/# apt install net-tools iputils-ping
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcap2-bin libpam-cap
The following NEW packages will be installed:
  iputils-ping libcap2-bin libpam-cap net-tools
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 339 kB of archives.
After this operation, 1313 kB of additional disk space will be used.
```

NB : noter bien l'id du conteneur

Dans un autre terminal, on crée une nouvelle image debian1

root@machine2:/home/elvis# docker ps -a

```
root@machine2:/home/elvis# docker ps -a
CONTAINER ID   IMAGE           COMMAND       CREATED         STATUS         PORTS     NAMES
b46a9326164d   debian:latest   "/bin/bash"   8 minutes ago   Up 8 minutes             quirky_tharp
root@machine2:/home/elvis#
```

root@machine2:/home/elvis# docker commit b46a9326164d debian1

```
root@machine2:/home/elvis#  docker commit b46a9326164d debian1
sha256:c640bbce7429448f999f06326992cb8c936730da974e3f41df1c3a6baa143d92
root@machine2:/home/elvis#
```

root@machine1:/home/elvis# docker ps –a

root@machine1:/home/elvis# docker commit e446841ba0fb debian1

```
root@machine1:/home/elvis# docker ps -a
CONTAINER ID   IMAGE           COMMAND       CREATED         STATUS         PORTS     NAMES
e446841ba0fb   debian:latest   "/bin/bash"   15 minutes ago   Up 15 minutes            unruffled_golick
root@machine1:/home/elvis#
root@machine1:/home/elvis# docker commit e446841ba0fb debian1
sha256:904bda1371b95b14c8192b52b06ca36d055e6d96006a51f4839135531a53274c
root@machine1:/home/elvis#
```

NB : A ce stade, debian est une image ayant les utilitaires réseaux ifconfig, ping

Créons sur la machine1 **docker1** et **docker2** sans les rattacher à un réseau dans un premier temps

root@machine1:/home/elvis# docker run -it --net=none --name docker1 debian1

```
root@machine1:/home/elvis# docker run -it --net=none --name docker1 debian1
root@1afa75ce7301:/#
root@1afa75ce7301:/#
```

root@machine1:/home/elvis# docker run -it --net=none --name docker2 debian1

```
root@machine1:/home/elvis# docker run -it --net=none --name docker2 debian1
root@24fb94f7f1f5:/#
root@24fb94f7f1f5:/#
```

Créons sur la machine2 **docker3** et **docker4**sans les rattacher à un réseau dans un premier temps

root@machine2:/home/elvis# docker run -it --net=none --name docker3 debian1

```
root@machine2:/home/elvis# docker run -it --net=none --name docker3 debian1
root@5ceaa53708d8:/#
root@5ceaa53708d8:/#
```

root@machine2:/home/elvis# docker run -it --net=none --name docker4 debian1

```
root@machine2:/home/elvis# docker run -it --net=none --name docker4 debian1
root@c14cb8b0c7c5:/#
root@c14cb8b0c7c5:/#
```

Ajoutons sur la machine 1 des adresses IP aux conteneurs 1 et 2 en utilisant l'outil ovs-docker

root@machine1:/home/elvis# sudo ovs-docker add-port ovs-br0 eth0 docker1 --ipaddress=192.168.1.11/24 --gateway=192.168.1.1

root@machine1:/home/elvis# sudo ovs-docker add-port ovs-br1 eth0 docker2 --ipaddress=192.168.2.11/24 --gateway=192.168.2.1

```
root@machine1:/home/elvis# sudo ovs-docker add-port ovs-br0 eth0 docker1 --ipaddress=192.168.1.11/24 --gateway=192.168.1.1
root@machine1:/home/elvis#
root@machine1:/home/elvis# sudo ovs-docker add-port ovs-br1 eth0 docker2 --ipaddress=192.168.2.11/24 --gateway=192.168.2.1
root@machine1:/home/elvis#
```

```
root@machine1:/home/elvis# docker run -it --net=none --name docker1 debian1
root@1afa75ce7301:/#
root@1afa75ce7301:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 66:2e:10:9c:eb:61  txqueuelen 1000  (Ethernet)
        RX packets 27  bytes 3314 (3.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@1afa75ce7301:/#
```

```
root@machine1:/home/elvis# docker run -it --net=none --name docker2 debian1
root@24fb94f7f1f5:/#
root@24fb94f7f1f5:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.11  netmask 255.255.255.0  broadcast 0.0.0.0
        ether ca:c7:10:e6:d8:cc  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 3384 (3.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@24fb94f7f1f5:/#
```

On fait de même sur la machine 2

root@machine2:/home/elvis# sudo ovs-docker add-port ovs-br0 eth0 docker3 --ipaddress=192.168.1.12/24 --gateway=192.168.1.1

root@machine2:/home/elvis# sudo ovs-docker add-port ovs-br1 eth0 docker4 --ipaddress=192.168.2.12/24 --gateway=192.168.2.1

```
root@machine2:/home/elvis# sudo ovs-docker add-port ovs-br0 eth0 docker3 --ipaddress=192.168.1.12/24 --gat
eway=192.168.1.1
root@machine2:/home/elvis#
root@machine2:/home/elvis# sudo ovs-docker add-port ovs-br1 eth0 docker4 --ipaddress=192.168.2.12/24 --gat
eway=192.168.2.1
root@machine2:/home/elvis#
root@machine2:/home/elvis#
```

```
root@machine2:/home/elvis# docker run -it --net=none --name docker3 debian1
root@5ceaa53708d8:/#
root@5ceaa53708d8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 92:9c:6a:5c:e5:49  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 2993 (2.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@5ceaa53708d8:/#
```

```
root@machine2:/home/elvis# docker run -it --net=none --name docker4 debian1
root@c14cb8b0c7c5:/#
root@c14cb8b0c7c5:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.12  netmask 255.255.255.0  broadcast 0.0.0.0
        ether d6:dd:69:2b:29:da  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 2993 (2.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@c14cb8b0c7c5:/#
```

- Configurer les Machines Virtuelles

root@machine1:/home/elvis# sudo ip addr add 10.0.1.43/24 dev enp0s3

root@machine1:/home/elvis# sudo ip link set dev enp0s3 up

```
root@machine1:/home/elvis# sudo ip addr add 10.0.1.43/24 dev enp0s3
root@machine1:/home/elvis# sudo ip link set dev enp0s3 up
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# ip addr show enp0s3

```
root@machine1:/home/elvis# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:59:56:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.32/16 brd 192.168.255.255 scope global dynamic enp0s3
       valid_lft 82007sec preferred_lft 82007sec
    inet 10.0.1.43/24 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::b64a:2074:9034:324b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
root@machine1:/home/elvis#
```

root@machine2:/home/elvis# sudo ip addr add 10.0.1.169/24 dev enp0s3

root@machine2:/home/elvis# sudo ip link set dev enp0s3 up

root@machine2:/home/elvis# ip addr show enp0s3

```
root@machine2:/home/elvis# sudo ip addr add 10.0.1.169/24 dev enp0s3
root@machine2:/home/elvis# sudo ip link set dev enp0s3 up
root@machine2:/home/elvis#
root@machine2:/home/elvis# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:5a:aa:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.179/16 brd 192.168.255.255 scope global dynamic enp0s3
       valid_lft 83320sec preferred_lft 83320sec
    inet 10.0.1.169/24 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe5a:aa17/64 scope link
       valid_lft forever preferred_lft forever
root@machine2:/home/elvis#
```

- **Tester un ping**

root@machine2:/home/elvis# ping 10.0.1.169

root@machine2:/home/elvis# ping 10.0.1.43

```
root@machine2:/home/elvis# ping 10.0.1.169
PING 10.0.1.169 (10.0.1.169) 56(84) bytes of data.
64 octets de 10.0.1.169 : icmp_seq=1 ttl=64 temps=0.045 ms
64 octets de 10.0.1.169 : icmp_seq=2 ttl=64 temps=0.097 ms
^C
--- statistiques ping 10.0.1.169 ---
2 paquets transmis, 2 reçus, 0 % paquets perdus, temps 1020 ms
rtt min/moy/max/mdev = 0,045/0,071/0,097/0,026 ms
root@machine2:/home/elvis#
root@machine2:/home/elvis# ping 10.0.1.43
PING 10.0.1.43 (10.0.1.43) 56(84) bytes of data.
64 octets de 10.0.1.43 : icmp_seq=1 ttl=64 temps=0.923 ms
64 octets de 10.0.1.43 : icmp_seq=2 ttl=64 temps=1.02 ms
^C
--- statistiques ping 10.0.1.43 ---
2 paquets transmis, 2 reçus, 0 % paquets perdus, temps 1002 ms
rtt min/moy/max/mdev = 0,923/0,971/1,019/0,048 ms
root@machine2:/home/elvis#
```

```
root@machine1:/home/elvis# ping 10.0.1.43
PING 10.0.1.43 (10.0.1.43) 56(84) bytes of data.
64 octets de 10.0.1.43 : icmp_seq=1 ttl=64 temps=0.054 ms
64 octets de 10.0.1.43 : icmp_seq=2 ttl=64 temps=0.089 ms
^C
--- statistiques ping 10.0.1.43 ---
2 paquets transmis, 2 reçus, 0 % paquets perdus, temps 1000 ms
rtt min/moy/max/mdev = 0,054/0,071/0,089/0,017 ms
root@machine1:/home/elvis# ping 10.0.1.169
PING 10.0.1.169 (10.0.1.169) 56(84) bytes of data.
64 octets de 10.0.1.169 : icmp_seq=1 ttl=64 temps=0.461 ms
64 octets de 10.0.1.169 : icmp_seq=2 ttl=64 temps=0.805 ms
^C
--- statistiques ping 10.0.1.169 ---
2 paquets transmis, 2 reçus, 0 % paquets perdus, temps 1025 ms
rtt min/moy/max/mdev = 0,461/0,633/0,805/0,172 ms
root@machine1:/home/elvis#
```

Étape 3 : établissement de tunnels VxLAN entre les deux machines

- SUR MACHINE 1

root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br0 vxlan0 -- set Interface vxlan0 type=vxlan options:remote_ip=10.0.1.169 options:key=1000

```
root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br0 vxlan0 -- set Interface vxlan0 type=vxlan options:remote_ip=10.0
.1.169 options:key=1000
root@machine1:/home/elvis#
```

- **Vérification de l'Interface vxlan0**

root@machine1:/home/elvis# sudo ovs-vsctl list Interface vxlan0

```
root@machine1:/home/elvis# sudo ovs-vsctl list Interface vxlan0
_uuid               : 1df07871-8071-437b-9c5c-99ae660e2e31
admin_state         : up
bfd                 : {}
bfd_status          : {}
cfm_fault           : []
cfm_fault_status    : []
cfm_flap_count      : []
cfm_health          : []
cfm_mpid            : []
cfm_remote_mpids    : []
cfm_remote_opstate  : []
duplex              : []
error               : []
external_ids        : {}
ifindex             : 15
ingress_policing_burst: 0
ingress_policing_rate: 0
lacp_current        : []
link_resets         : 0
link_speed          : []
link_state          : up
lldp                : {}
mac                 : []
mac_in_use          : "2a:22:47:19:cb:89"
mtu                 : []
mtu_request         : []
name                : vxlan0
ofport              : 3
ofport_request      : []
options             : {key="1000", remote_ip="10.0.1.169"}
other_config        : {}
statistics          : {rx_bytes=0, rx_packets=0, tx_bytes=0, tx_packets=0}
status              : {tunnel_egress_iface=enp0s3, tunnel_egress_iface_carrier=up}
type                : vxlan
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br1 vxlan1 -- set Interface vxlan1 type=vxlan options:remote_ip=10.0.1.169 options:key=2000

```
root@machine1:/home/elvis# sudo ovs-vsctl add-port ovs-br1 vxlan1 -- set Interface vxlan1 type=vxlan options:remote_ip=10.0
.1.169 options:key=2000
root@machine1:/home/elvis#
root@machine1:/home/elvis#
```

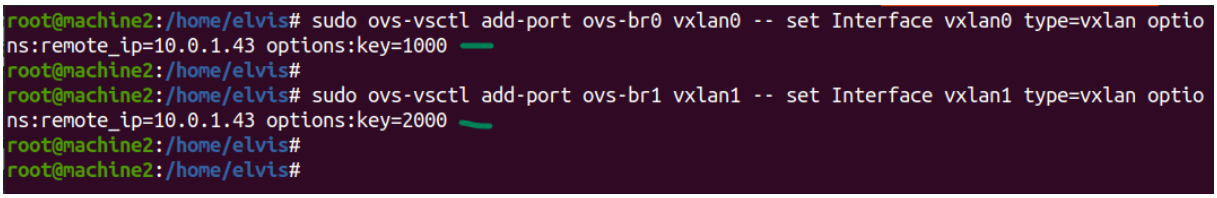
- **Vérification de l'Interface vxlan1**

root@machine1:/home/elvis# sudo ovs-vsctl list Interface vxlan1

- SUR MACHINE 2

root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br0 vxlan0 -- set Interface vxlan0 type=vxlan options:remote_ip=10.0.1.43 options:key=1000

root@machine2:/home/elvis# sudo ovs-vsctl add-port ovs-br1 vxlan1 -- set Interface vxlan1 type=vxlan options:remote_ip=10.0.1.43 options:key=2000



root@machine2:/home/elvis# sudo ovs-vsctl list Interface vxlan1

root@machine2:/home/elvis# sudo ovs-vsctl list Interface vxlan0

```
root@machine2:/home/elvis# sudo ovs-vsctl list Interface vxlan0
_uuid               : 89c4b2e3-e1e9-4daa-8c0f-858412290dab
admin_state         : up
bfd                 : {}
bfd_status          : {}
cfm_fault           : []
cfm_fault_status    : []
cfm_flap_count      : []
cfm_health          : []
cfm_mpid            : []
cfm_remote_mpids    : []
cfm_remote_opstate  : []
duplex              : []
error               : []
external_ids        : {}
ifindex             : 5
ingress_policing_rate: 0
lacp_current        : []
link_resets         : 0
link_speed          : []
link_state          : up
lldp                : {}
mac                 : []
mac_in_use          : "a2:e2:1c:95:84:26"
mtu                 : []
mtu_request         : []
name                : vxlan0
ofport              : 3
ofport_request      : []
options             : {key="1000", remote_ip="10.0.1.43"}
other_config        : {}
statistics          : {rx_bytes=0, rx_packets=0, tx_bytes=0, tx_packets=0}
status              : {}
type                : vxlan
root@machine2:/home/elvis#
```

On démarre sur la machine 1 docker1 et docker2

root@machine1:/home/elvis# docker start docker1

root@machine1:/home/elvis# docker start docker2

```
root@machine1:/home/elvis# docker start docker1
docker1
root@machine1:/home/elvis# docker start docker2
docker2
root@machine1:/home/elvis#
```

On démarre sur la machine 2 docker3 et docker4

root@machine2:/home/elvis# docker start docker3

root@machine2:/home/elvis# docker start docker4

```
root@machine2:/home/elvis# docker start docker3
docker3
root@machine2:/home/elvis#
root@machine2:/home/elvis# docker start docker4
docker4
root@machine2:/home/elvis#
```

- **Étape 4 : test de connectivité entre les machines de même vlan se trouvant sur 2 machines distantes**

De docker1 vers docker3

Le ping devrait marcher si tout se passe bien

root@machine1:/home/elvis# docker exec docker1 ping 192.168.1.12

```
root@machine1:/home/elvis# docker exec docker1 ping 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.
64 bytes from 192.168.1.12: icmp_seq=1 ttl=64 time=1.60 ms
64 bytes from 192.168.1.12: icmp_seq=2 ttl=64 time=0.766 ms
64 bytes from 192.168.1.12: icmp_seq=3 ttl=64 time=0.663 ms
^C
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# docker exec docker1 ping 192.168.1.11

```
root@machine1:/home/elvis# docker exec docker1 ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.082 ms
^C
root@machine1:/home/elvis#
```

root@machine2:/home/elvis# docker exec docker3 ping 192.168.1.11

```
root@machine2:/home/elvis#  docker exec docker3 ping 192.168.1.11
PING 192.168.1.11 (192.168.1.11) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.595 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.427 ms
^C
root@machine2:/home/elvis#
```

root@machine2:/home/elvis# docker exec docker3 ping 192.168.1.12

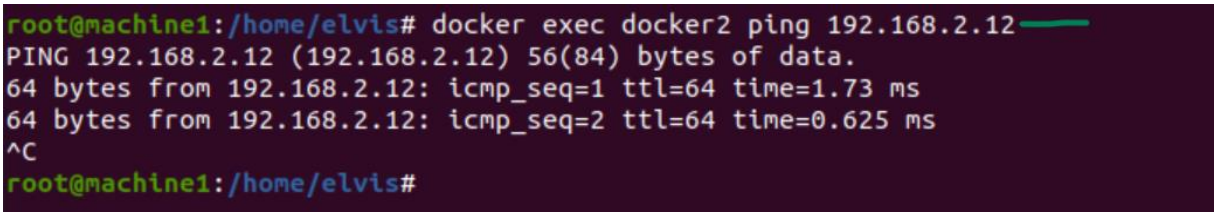
```
root@machine2:/home/elvis# docker exec docker3 ping 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.
64 bytes from 192.168.1.12: icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from 192.168.1.12: icmp_seq=2 ttl=64 time=0.078 ms
^C
root@machine2:/home/elvis#
```

De docker2

root@machine1:/home/elvis# docker exec docker2 ping 192.168.2.11

```
root@machine1:/home/elvis# docker exec docker2 ping 192.168.2.11
PING 192.168.2.11 (192.168.2.11) 56(84) bytes of data.
64 bytes from 192.168.2.11: icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from 192.168.2.11: icmp_seq=2 ttl=64 time=0.076 ms
^C
root@machine1:/home/elvis#
```

root@machine1:/home/elvis# docker exec docker2 ping 192.168.2.12



CONCLUSION GENERALE

Lorsqu'on combine le **SDN** et le **VXLAN**, nous assistons à une évolution radicale des réseaux modernes, où la flexibilité, l'évolutivité et l'automatisation prennent une place centrale. Le SDN, avec sa séparation du contrôle et des données, et sa capacité à automatiser la gestion des réseaux, offre aux entreprises un outil puissant pour s'adapter rapidement aux besoins croissants des infrastructures complexes, notamment avec l'essor des technologies comme la 5G et l'IoT. D'autre part, VXLAN étend cette flexibilité en permettant de surmonter les limitations des VLANs traditionnels, en créant des réseaux virtuels massivement évolutifs, capables de s'étendre au-delà des frontières physiques des datacenters.

Notons que ces deux technologies se complètent parfaitement dans la construction des réseaux du futur : tandis que le SDN centralise et automatise la gestion des flux de données, VXLAN garantit une isolation et une extensibilité optimales dans les environnements virtualisés et multi-tenant. Ensemble, elles permettent de créer des réseaux plus agiles, évolutifs et performants, en répondant aux défis posés par la croissance des besoins en connectivité dans un monde de plus en plus tourné vers le cloud et les infrastructures décentralisées.