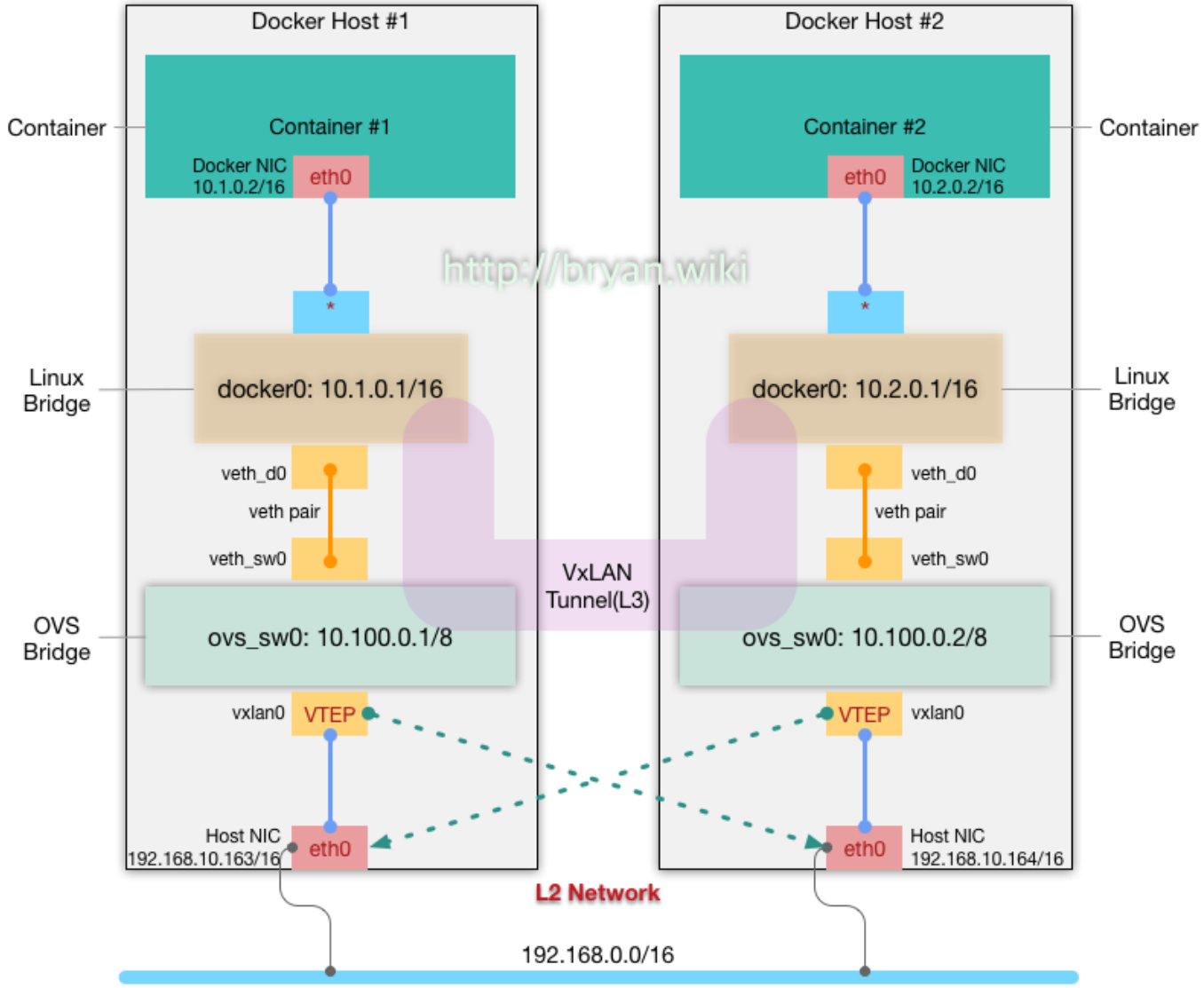
목차

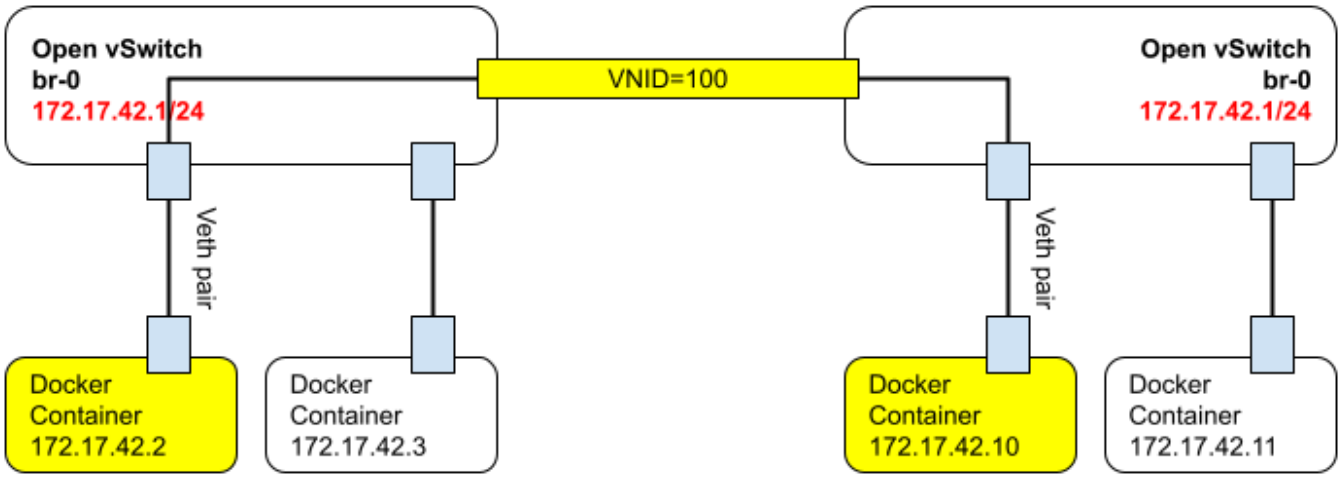
openvswitch를 통한 vxlan 구성 3
사전 준비 4
Open vSwitch 설치와 firewall 설정 4
접속 경로의 연결을 위한 Open vSwitch Bridge와 VxLAN 설정 5
Linux Bridge와 Open vSwitch Bridge 간의 연결선, Veth Pair 6
libvirt network 추가 7
NetworkManager-ovs 를 통한 vxlan 구성 7
linux network 를 이용한 설정 8
참고문서 8
NetworkManager-ovs 8

openvswitch를 통한 vxlan 구성

출처: https://bryan.wiki/277 [Bryan's Tech-Log]



그림을 보면, 2개의 호스트가 Flat한 L2 Network로 연결되어 있고(SDN: Software-Defined-Network 개념으로 보면 Underlay) 각각의 호스트 내에 Docker Container 들은 서로 다른 네트워크를 가지며, VxLAN 터널을 통한 L3 연결(SDN 개념으로 보면 Overlay)가 가능한 구조임을 알 수 있다. 그림과 같은 네트워크를 설계할 때에 특히 주의해야 할 점은, 하단의 OVS Bridge의 네트워크(10.0.0.0/8)가 상단의 Linux Bridge(docker0)의 네트워크(10.x.0.0/16)를 포함하는 구조를 가져야 한다는 것이다



첫번째 그림과 다른점은 ovs bridge의 IP가 서로 다를 필요가 없다는것이다. 즉 GW주소는 양쪽에 10.100.0.1 로 동일하게 구성해도 상관없다는것이다.

사전 준비

- Flat Network로 연결된 Docker Host(VM) 2대
- Docker Host #1: CentOS 7.3 Minimal Server(1611 ver), 192.168.10.163/16
- Docker Host #2: CentOS 7.3 Minimal Server(1611 ver), 192.168.10.164/16

Open vSwitch 설치와 firewall 설정

- openvswitch 수동 설치는 CentOS 7 에 ovs 설치 문서를 참고하도록 한다.

```
yum clean all
yum install -y epel-release https://www.rdoproject.org/repos/rdo-release.rpm
yum install -y firewalld docker openvswitch bridge-utils
yum update -y
systemctl start openvswitch firewalld
systemctl enable openvswitch firewalld
```

- VxLAN 터널링을 위한 패킷은 UDP 4789, 8472 포트를 통해 전송된다. firewall 설정에서 이 2개 포트를 모두 개방

```
firewall-cmd --add-port=4789/udp --add-port=8472/udp
firewall-cmd --permanent --add-port=4789/udp --add-port=8472/udp
```

- trusted 영역에 대해 10.*.*.* 대역의 통신이 모두 가능하도록 개방(docker0 네트워크 10.x.0.0/16 을 서브넷으로 포함하는 상위 네트워크의 예: 10.0.0.0/8)
- 다른 예: docker0 네트워크를 172.17.0.0/16 이나 172.18.0.0/16 … 등으로 할 경우에는 상위 네트워크를 172.16.0.0/12 로 하면 된다

```
firewall-cmd --permanent --zone=trusted --add-source=10.0.0.0/8
```

접속 경로의 연결을 위한 Open vSwitch Bridge와 VxLAN 설정

- docker01 에서는 다음과 같이 Open vSwitch 를 설정

```
[root@docker01 ~#] ovs-vsctl add-br ovs_sw0

[root@docker01 ~#] ip addr add 10.100.0.1/8 dev ovs_sw0 && ip link set dev
ovs_sw0 up

[root@docker01 ~#] ovs-vsctl add-port ovs_sw0 vxlan0 -- set Interface vxlan0
type=vxlan options:remote_ip=192.168.10.164
```

- docker02 에서는 다음과 같이 Open vSwitch 를 설정

```
[root@docker02 ~#] ovs-vsctl add-br ovs_sw0

[root@docker02~#] ip addr add 10.100.0.2/8 dev ovs_sw0 && ip link set dev
ovs_sw0 up

[root@docker02 ~#] ovs-vsctl add-port ovs_sw0 vxlan0 -- set Interface vxlan0
type=vxlan options:remote_ip=192.168.10.163
```

위의 그림에서 보듯이 각 Docker 호스트에 sw0 라는 Open vSwitch 브리지간의 터널을 개통하는 과정이다. 주의 깊게 보아야 하는 부분은 vxlan0 라는 VTEP(VxLAN Terminal End Point)가 상대편 Docker 호스트의 네트워크 IP를 바라보게 설정하는 것이다.

여기까지 설정하였다면 그림에서의 하단(Open vSwitch 브리지간)의 터널 개통이 정상적으로 되었는지 확인해 보기로 하자

```
[root@docker01 ~]# ping 10.100.0.2
PING 10.100.0.2 (10.100.0.2) 56(84) bytes of data.
64 bytes from 10.100.0.2: icmp_seq=1 ttl=64 time=2.43 ms
64 bytes from 10.100.0.2: icmp_seq=2 ttl=64 time=0.418 ms
64 bytes from 10.100.0.2: icmp_seq=3 ttl=64 time=0.333 ms
64 bytes from 10.100.0.2: icmp_seq=4 ttl=64 time=0.354 ms
64 bytes from 10.100.0.2: icmp_seq=5 ttl=64 time=0.410 ms
^C

--- 10.100.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.333/0.790/2.438/0.824 ms
```

10.100.0.1/8 과 10.100.0.2/8 은 동일한 네트워크 대역이고, VxLAN 터널을 통해 서로 연결이 잘 되고 있음을 확인할 수 있다.

Linux Bridge와 Open vSwitch Bridge 간의 연결선, Veth Pair

하단의 터널이 뚫렸으므로 상단의 연결선을 만들어 보자. Docker container(브리지 모드일 경우)는 기본적으로 Linux Bridge와 자동으로 연결이 맺어 진다. 즉, Docker 엔진이 docker0 라는 디폴트 브리지(Linux Bridge)에 Docker 컨테이너의 네트워크 디바이스를 내부적으로 연결시켜 주게 된다.

그렇다면 우리는, 그 아래의 docker0와 sw0 사이의 연결을 맺어 주기만 하면 되는데, 이를 위해 VETH pair 라고 하는, 양쪽 끝이 연결되어 있는 연결 쌍(pair)이라는 도구를 사용해서 위 아래의 접점에 붙여 주면 된다.

그림에서 처럼, docker0(Linux Bridge)와 sw0(Open vSwitch Bridge)간의 연결을, Veth Pair 라고 하는 연결선을 통해서 맺어 줄 수 있다. 다음과 같이 해 보자. 이 연결선의 End Point(끝점) 들 중에서 veth_d0는 docker0 쪽으로, veth_sw0는 sw0 쪽으로 연결한다.

```
[root@docker01 ~]# ip link add veth_sw0 type veth peer name veth_d0
[root@docker01 ~]# ovs-vsctl add-port ovs_sw0 veth_sw0
[root@docker01 ~]# brctl addif docker0 veth_d0
[root@docker01 ~]# ip link set dev veth_sw0 up
[root@docker01 ~]# ip link set dev veth_d0 up
```

docker02 에서도 위와 같이 설정해 주자. End Point 들의 이름은 docker01 에서와 다르게 주어도 무관하지만 script 작성 등의 자동화를 위해서는 최대한 일관성 유지를 하는 것이 좋다.

```
[root@docker02 ~]# ip link add veth_sw0 type veth peer name veth_d0
[root@docker02 ~]# ovs-vsctl add-port ovs_sw0 veth_sw0
[root@docker02 ~]# brctl addif docker0 veth_d0
[root@docker02 ~]# ip link set dev veth_sw0 up
[root@docker02 ~]# ip link set dev veth_d0 up
```

각각의 Docker 호스트에서 다음과 같이 실행해서 브리지 설정을 확인한다.

```
[root@docker01 ~]# ovs-vsctl show
f3c8825d-73ba-4ee5-a136-3db14a32e990
    Bridge "ovs_sw0"
        Port "veth_sw0"
            Interface "veth_sw0"
        Port "ovs_sw0"
            Interface "ovs_sw0"
                type: internal
        Port "vxlan0"
            Interface "vxlan0"
                type: vxlan
                options: {remote_ip="192.168.10.164"}
    ovs_version: "2.5.2"
```

```
[root@docker01 ~]# brctl show
bridge name bridge id   STP enabled interfaces
docker0   8000.0242f84de852 no    veth_d0
```

libvirt network 추가

kvm guest에 할당할 vxlan을 이용한 Virtual Network를 생성한다.

아래와 같은 vxnet0.xml 파일을 생성한다.

```
<network>
  <name>vxnet0</name>
  <forward mode='bridge'/>
  <bridge name='ovs-sw0'/>
  <virtualport type='openvswitch'/>
</network>
```

생성한 xml을 이용하여 libvirt Virtual Network를 생성한다.

```
virsh net-define vxnet0.xml

virsh net-start vxnet0
virsh net-autostart vxnet0
```

이후 GuestVM에 해당 vxnet0 네트워크 인터페이스를 할당하면 된다.

NetworkManager-ovs 를 통한 vxlan 구성

```
nmcli conn add type ovs-bridge conn.interface ovs-sw0
nmcli conn add type ovs-port conn.interface ovs-sw0 master ovs-sw0
nmcli conn add type ovs-interface slave-type ovs-port conn.interface ovs-sw0
master ovs-sw0 ipv4.method manual ipv4.address 10.100.0.1/24

ovs-vsctl add-port ovs-sw0 vxlan0 -- set Interface vxlan0 type=vxlan
options:remote_ip=192.168.0.102
```

위 예제에서도 보이듯이 `ovs-bridge`와 `ovs-port`, `ovs-interface` 모든 interface의 name을 동일하게 맞춰줘야 한다. 그렇지 않다면 kvm virtual network 에서 정확한 정보를 가져오지 못해 vm의 network interface port 할당이 제대로 되지 않아 부팅에 실패하게 된다.

linux network 를 이용한 설정

영구적으로 적용하려면 linux network script로 작성하면 된다.

```
[root@kvm31 network-scripts]# vim /etc/sysconfig/network-scripts/ifcfg-
ovsbr0
DEVICE=ovsbr0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPV6INIT=no
DELAY=0
IPADDR=10.10.10.1
PREFIX=8
HOTPLUG=no
```

만약 포트를 추가하고 싶은경우 아래와 같이 인터페이스를 생성한다. 아래는 물리NIC를 포트에 연결하는 예제이다.

```
[root@kvm31 network-scripts]# vim /etc/sysconfig/network-scripts/ifcfg-
eno16777736
HWADDR="00:0C:29:62:F9:D3"
TYPE="OVSPort"
DEVICETYPE="ovs"
OVS_BRIDGE="ovsbr0"
BOOTPROTO="none"
NAME="eno16777736"
UUID="5cc31ab7-c26d-48b6-89f2-a7c933d53cb8"
ONBOOT="yes"
NM_CONTROLLED="no"
```

참고문서

- https://bryan.wiki/277
- https://bryan.wiki/279
- https://www.joinc.co.kr/w/man/12/OpenVSwitch/VXLAN
- https://blog.christophersmart.com/2020/07/27/how-to-create-linux-bridges-and-open-vswitch-bridges-with-networkmanager/
- https://blog.oddbit.com/post/2020-02-15-configuring-open-vswitch-with/

NetworkManager-ovs

```
# subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-4.6-rpms" \
    --enable="rhel-7-server-optional-rpms" \    <--- Add for
"NetowrkManager-ovs" package
    --enable="rhel-7-fast-datapath-rpms"        <--- Add for
"openvswitch2.13" package
```

- https://developer.gnome.org/NetworkManager/stable/nm-openvswitch.html
- https://blog.oddbit.com/post/2020-02-15-configuring-open-vswitch-with/

From:
https://atl.kr/dokuwiki/ - **AllThatLinux!**

Permanent link:
https://atl.kr/dokuwiki/doku.php/openvswitch%EB%A5%BC_%ED%86%B5%ED%95%9C_vxlan_%EA%B5%AC%EC%84%B1?rev=1625449850

Last update: **2021/07/05 01:50**