Open vSwitch – lab

Radosław Kujawa – radoslaw.kujawa@osec.pl

OSEC

14 czerwca 2017

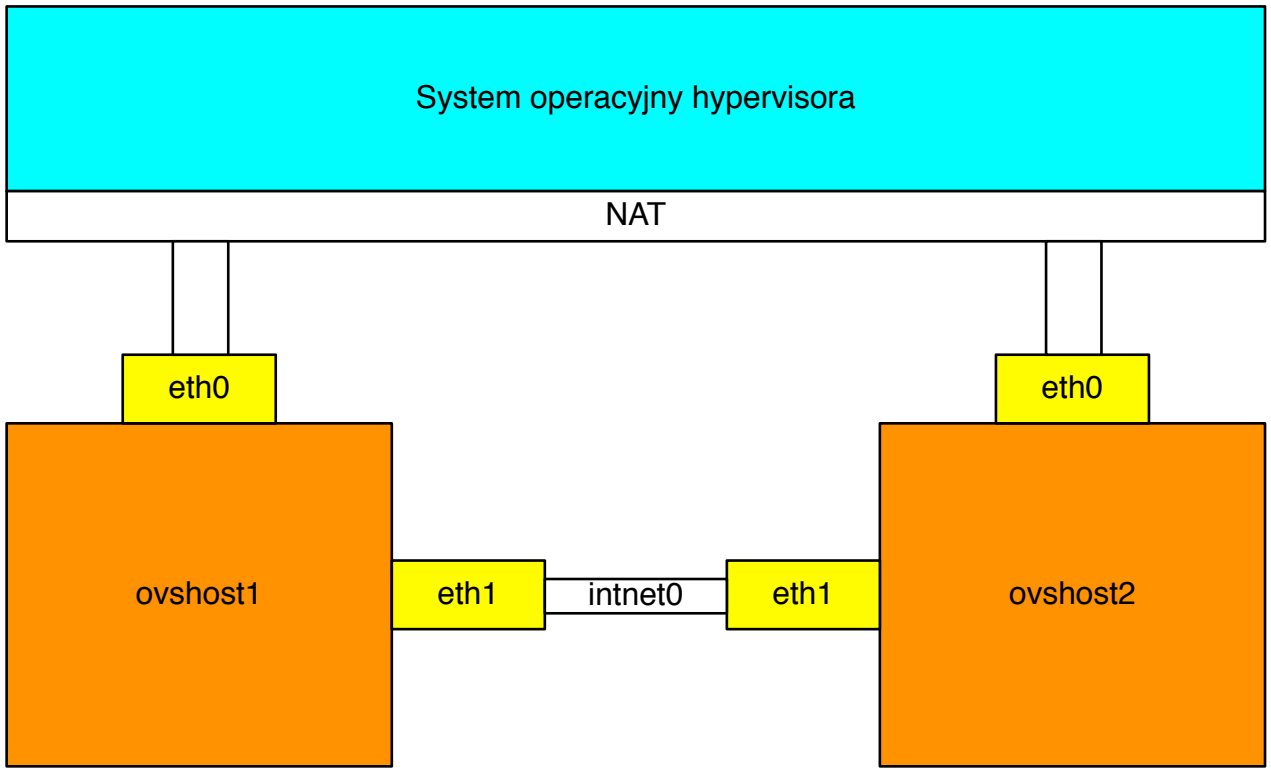
Open vSwitch

- Elastyczny, zarządzalny wirtualny switch do zastosowań Software Defined Network.
- Do zarządzania wykorzystywany jest protokół OpenFlow.
- Używany domyślnie do wirtualizacji warstwy 2 sieci w Red Hat OpenStack.
- Do separacji warstwy drugiej można wykorzystać szereg mechanizmów (VLAN/VXLAN/GRE...).

Przygotowania do instalacji

- Open vSwitch dostępny w większości dystrybucji Linuxa.
- W Fedorze paczka `openvswitch`.
- Wymagania dotyczące konfiguracji.
  - NetworkManager konfliktuje z Open vSwitch.
  - `systemctl stop NetworkManager`
  - `systemctl disable NetworkManager`
  - W naszym obrazie VM jest już wyłączony.

Nasze środowisko demonstracyjne



Środowisko demonstracyjne – import maszyn

- Obraz maszyn wirtualnych w formacie OVF 2.0.
- Reinitialize the MAC address of all network cards.
- Fedora 26 Alpha + Open vSwitch 2.7.0.
- Logowanie: `root/fedora`
- Nadanie nazwy hosta:
- `hostnamectl set-hostname ovshostX.osecforum.pl`
- Materiały do labu:
- `cd && git clone` https://github.com/OSEC-pl/osecforum-openvswitch.git `lab`

Środowisko demonstracyjne – firewall

- W rzeczywistych warunkach konieczna jest także konfiguracja firewalla (ale tu nie będziemy tego robić).
- `systemctl stop firewalld`
- `systemctl disable firewalld`
- `systemctl mask firewalld`

Środowisko demonstracyjne – konfiguracja

- Przkierowanie połączenia SSH do maszyn wirtualnych:
  - ovshostX: Network settings ▷ Adapter 1 ▷ Port Forwarding.
  - Host IP: 127.0.0.1
  - Host port: 1000X
  - Guest IP: 10.0.2.15
  - Guest port: 22
  - Można zalogować się po SSH z systemu operacyjnego hypervisora:
  - `ssh -p 1000X root@127.0.0.1`
- Zezwolenie na przechywywanie ruchu w sieci wewnętrznej:
  - ovshostX: Network settings ▷ Adapter 2 ▷ Promiscuous Mode.
  - Allow all.

SELinux

- Fedora 26 Alpha wymaga małych poprawek do polityki bezpieczeństwa SELinuxa.
- `semodule -i $HOME/lab/selinux/ovs-f26-fix.pp`

Usługa Open vSwitch

- Paczka: `openvswitch`.
- Jednostka systemd: `openvswitch.service` (podnosi też `ovs-vswitchd.service` i `ovsdb-server.service`).
- Moduł kernela: `openvswitch`.
- `systemctl start openvswitch`
- `systemctl enable openvswitch`

Podstawowa konfiguracja bridge'a

- `ovs-vsctl add-br ovsbr0`
- `ovs-vsctl add-port ovsbr0 eth1`
- `ovs-vsctl show`
- `ip addr add 172.24.254.X/24 dev ovsbr0`
- `ip link set dev ovsbr0 up`

```
Bridge "ovsbr0"
  Port "eth1"
    Interface "eth1"
  Port "ovsbr0"
    Interface "ovsbr0"
    type: internal
ovs_version: "2.7.0"
```

Użycie VLAN do separacji sieci warstwy 2

- Wszystkie porty w Open vSwitch domyślnie są trunkami VLAN.
- Sytuacja z poprzedniego slajdu – ruch „nieotagowany” (bez VLAN).
- Dodanie interfejsu w konkretnym VLAN:
  - `ovs-vsctl add-port ovsbr0 vlan1 -- set interface vlan1 type=internal`
  - `ovs-vsctl set port vlan1 tag=1`
  - `ip addr add 172.24.253.X/24 dev vlan1`
  - `ip link set dev vlan1 up`

Użycie VLAN do separacji sieci warstwy 2 – c.d.

- Port `vlan1` jest widoczny na poziomie hosta jako osobny interfejs.
- Ruch wysłany/odbierany na tym interfejsie jest automatycznie tagowany jako VLAN z ID 1.

```
Bridge "ovsbr0"
  Port "vlan1"
    tag: 1
    Interface "vlan1"
      type: internal
  Port "eth1"
    Interface "eth1"
  Port "ovsbr0"
    Interface "ovsbr0"
      type: internal
ovs_version: "2.7.0"
```

Tunelowanie ruchu w sieci fizycznej

- Użycie VLAN do separacji na poziomie sieci fizycznej nie zawsze jest możliwe.
- Open vSwitch pozwala tunelować ruch między bridge'ami na odległych maszynach.
- Konfigurujemy adresację IP eth1 na poziomie systemu operacyjnego.
- `ovs-vsctl del-port ovsbr0 eth1`
- `ip addr add 192.168.1.X/24 dev eth1`
- Stworzenie tunelu GRE:
- `ovs-vsctl add-port ovsbr0 ovsgre0 -- set interface ovsgre0 type=gre options:remote_ip=192.168.1.Y`

Tunelowanie ruchu z użyciem VXLAN

- Protokół GRE, jako odrębny protokół IP nie zawsze jest preferowanym rozwiązaniem.
- Protokół VXLAN jest bardziej efektywnym rozwiązaniem gdy mamy wiele hostów – tunelowanie ramek L2 w UDP.
- VXLAN wspiera też multicasting... choć sam Open vSwitch jeszcze nie. Implementacja w kernelu Linuxa ma już obsługę multicastingu.
- Konfiguracja tunelu VXLAN w Open vSwitch:
- `ovs-vsctl add-port ovsbr0 ovsvx0 -- set interface ovsvx0 type=vxlan options:remote_ip=192.168.1.Y`

Network namespaces

- Pozwala na uruchomienie więcej niż jednej „instancji” stosu sieciowego w kernelu Linuxa.
- Każdy namespace ma własne, odrębne interfejsy, własną konfigurację adresacji i tablicę routingu.
- Pozwalają na zwirtualizowanie funkcji sieciowych wymagających dostępu do warstwy 2.
- Dodanie namespace:
- `ip netns add foo`
- `ip netns list`
- Wykonanie polecenia wewnątrzn namespace:
- `ip netns exec foo polecenie`
- `ip netns exec foo ip link set dev lo up`

Interfejsy `veth`

- Wirtualny Ethernet – posiada dwie „końcówki”.
- Może posłużyć do połączenia namespace z Open vSwitch.
- Utworzenie interfejsów `veth`:
- `ip link add veth0 type veth peer name veth1`
- `ip link set veth1 netns foo`
- `ip netns exec foo ip link set dev veth1 up`
- `ip netns exec foo ip addr add 172.24.252.X/24 dev veth1`

Dodanie veth do Open vSwitch jako portu

- `ip link set dev veth0 up`
- `ovs-vsctl add-port ovsbr0 veth0`

```
Bridge "ovsbr0"
  Port "ovsvx0"
    Interface "ovsvx0"
      type: vxlan
      options: {remote_ip="192.168.1.2"}
    Port "veth0"
      tag: 2
      Interface "veth0"
```

- `ip netns exec foo ping 172.24.252.Y`
- `ip netns exec foo ping6 fe80::...%veth1`

Stała konfiguracja

- Konfiguracja bridge'y i wewnętrznych portów z założenia jest stała, konfiguracja adresacji IP *nie jest*.
- Gdy nie mamy kontrolera SDN, możemy skonfigurować Open vSwitcha przy starcie za pomocą skryptów w katalogu `/etc/sysconfig/network-scripts/`.
- OpenStack składa w ten sposób tylko bridge `br-ex`, reszta jest konfigurowana agentem Neutrona.
- Nie wszystkie możliwe rodzaje konfiguracji są wspierane.
- Należy pamiętać o odpowiedniej konfiguracji firewalla.

Stała konfiguracja c.d.

- Przykład: fizyczny interfejs do Open vSwitcha.

```
# ifcfg-ovsbr0
DEVICE=ovsbr0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=X.X.X.X
PREFIX=XX
```

```
# ifcfg-eth1
DEVICE=eth1
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSPort
BOOTPROTO=none
OVS_BRIDGE=ovsbr0
```

Stała konfiguracja c.d.

- Przykład: enkapsulacja sieci L2 w protokół GRE.

```
# ifcfg-ovsbr0
DEVICE=ovsbr0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=X.X.X.X
PREFIX=XX
```

```
# ifcfg-ovsgre0
DEVICE=ovsgre0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSTunnel
OVS_BRIDGE=ovsbr0
OVS_TUNNEL_TYPE=gre
OVS_TUNNEL_OPTIONS=
"options:remote_ip=Y.Y.Y.Y"
```

- Ponadto standardowe interfejsy ethernetowe w sieci, po której przesyłany jest tunel.

Stała konfiguracja c.d.

- Wirtualizacja – interfejsy tap maszyn wirtualnych (`vnetX` w `libvirtd`). Tak samo jak każdy inny interfejs tam w Open vSwitch.
- Network namespace – brak standardu dla statycznej konfiguracji.
- Co z `veth`? https://github.com/LanetNetwork/initscripts-veth (dla Linux Bridge, potrzebne poprawki dla Open vSwitch).
- Pole dla własnej inwencji twórczej...
- W środowisku korporacyjnym lepiej użyć kontrolera SDN – niech on konfiguruje Open vSwitcha.

Podsumowując...

- Open vSwitch w połączeniu z namespace oraz interfejsami `veth` daje nam niespotykane wcześniej możliwości wirtualizacji sieci oraz funkcji sieciowych.
- Pozwala na łatwe wdrożenie aplikacji wymagających bezpośrednio dostępu do warstwy drugiej sieci.
- Rozwiązanie to chętnie jest wykorzystywane przy tworzeniu chmur prywatnych (np. wirtualne routery, usługi sieciowe w OpenStack).

Koniec...

Dziękuje!
Czy są pytania?