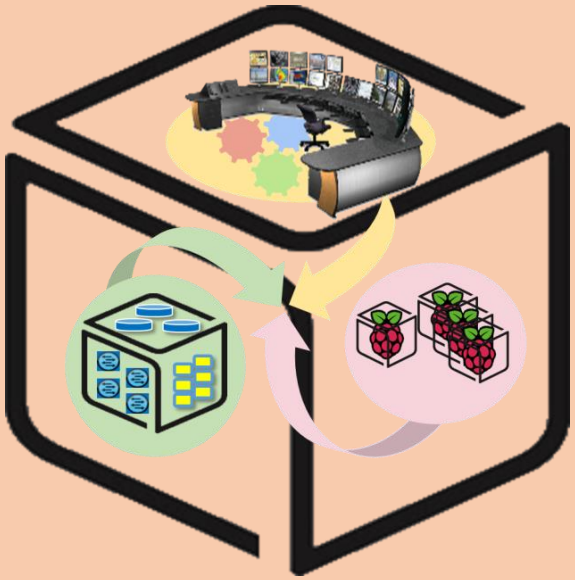
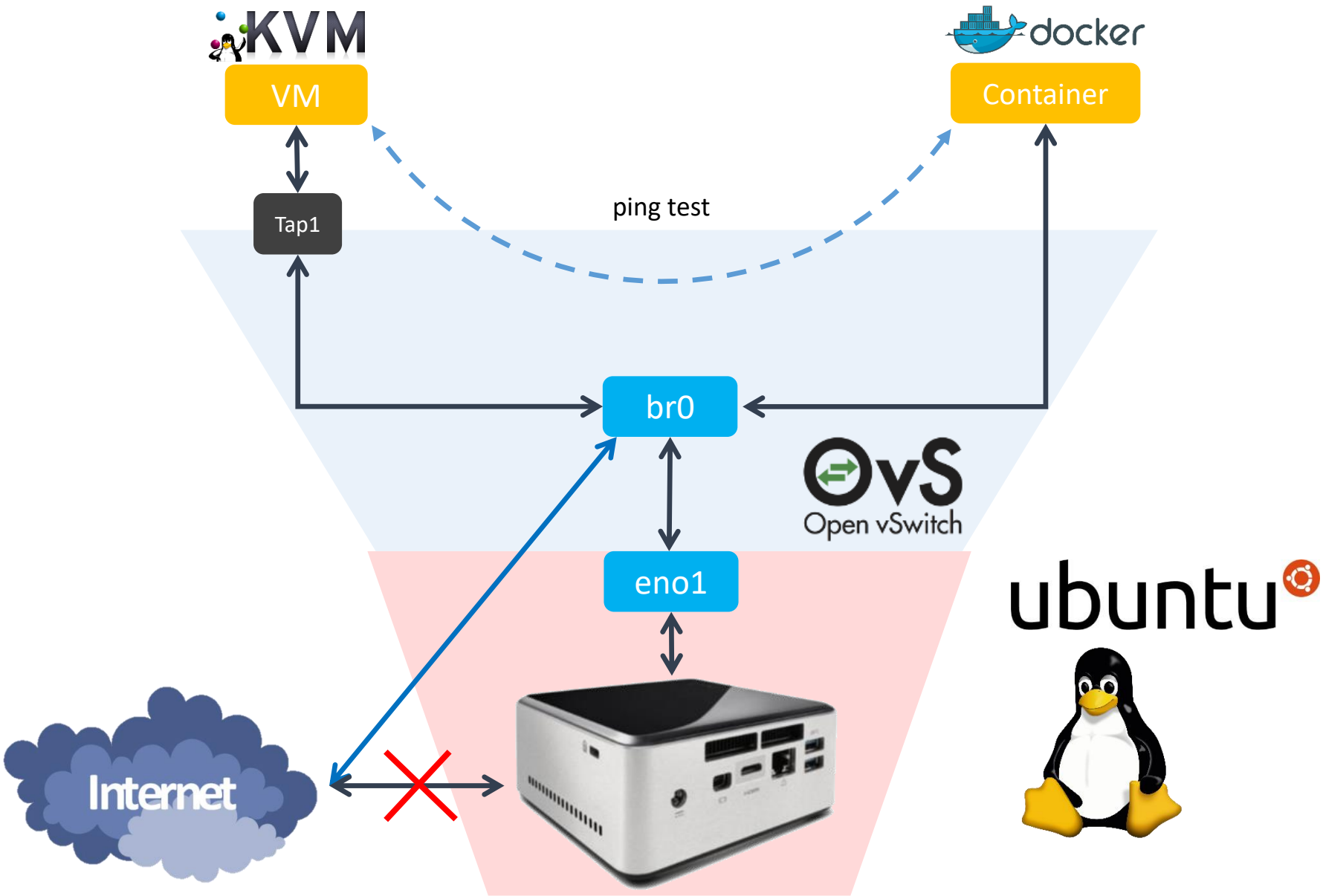
Computer Systems For AI-inspired Cloud Theory & Lab.

Lab #1: Box

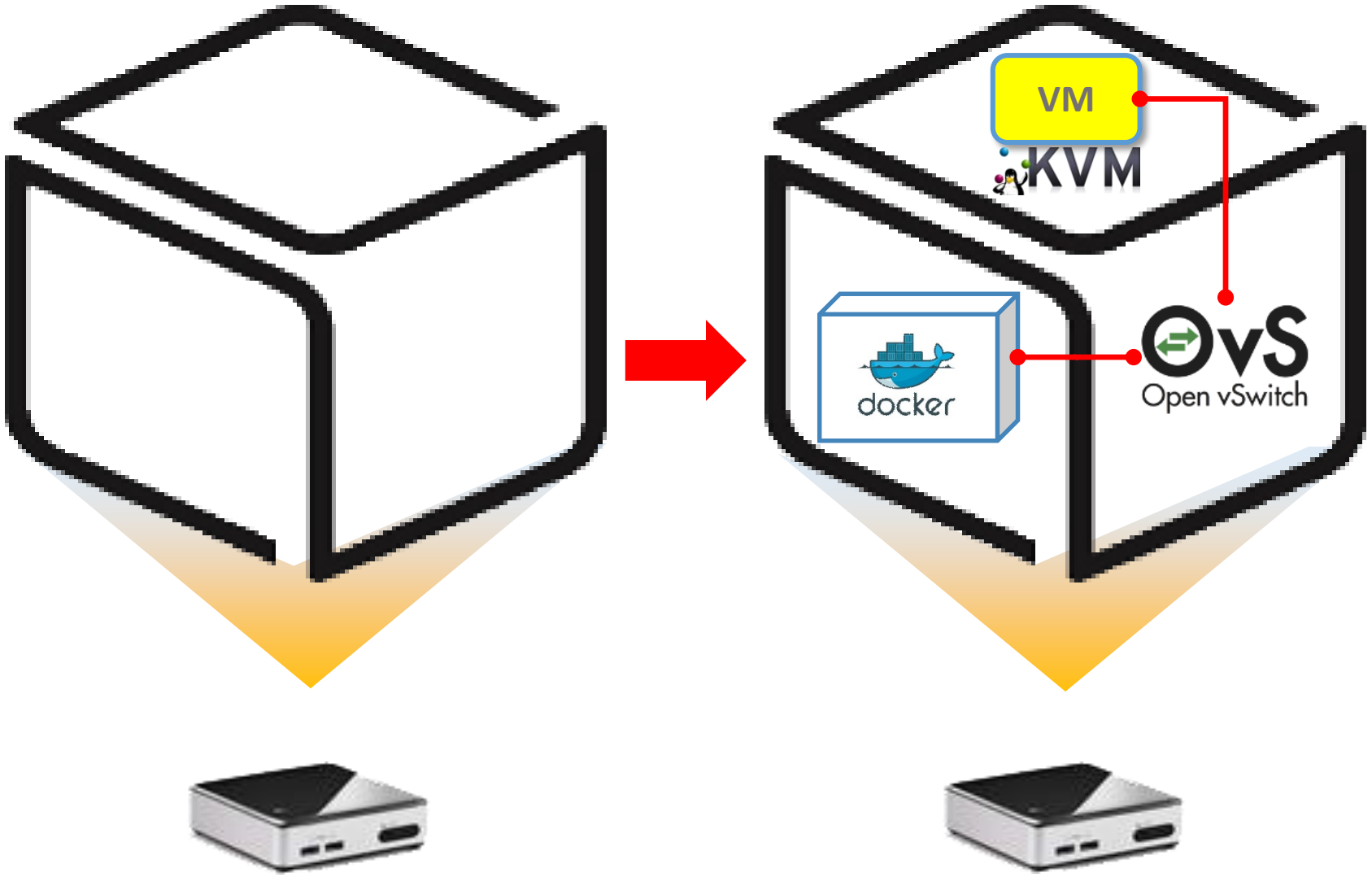


https://github.com/SmartX-Labs/SmartX-Mini-MOOC

Box Lab: Outline



Box Lab: Final Goal



Before you start

- Things you need to know

Lab Theory

Lab Practice

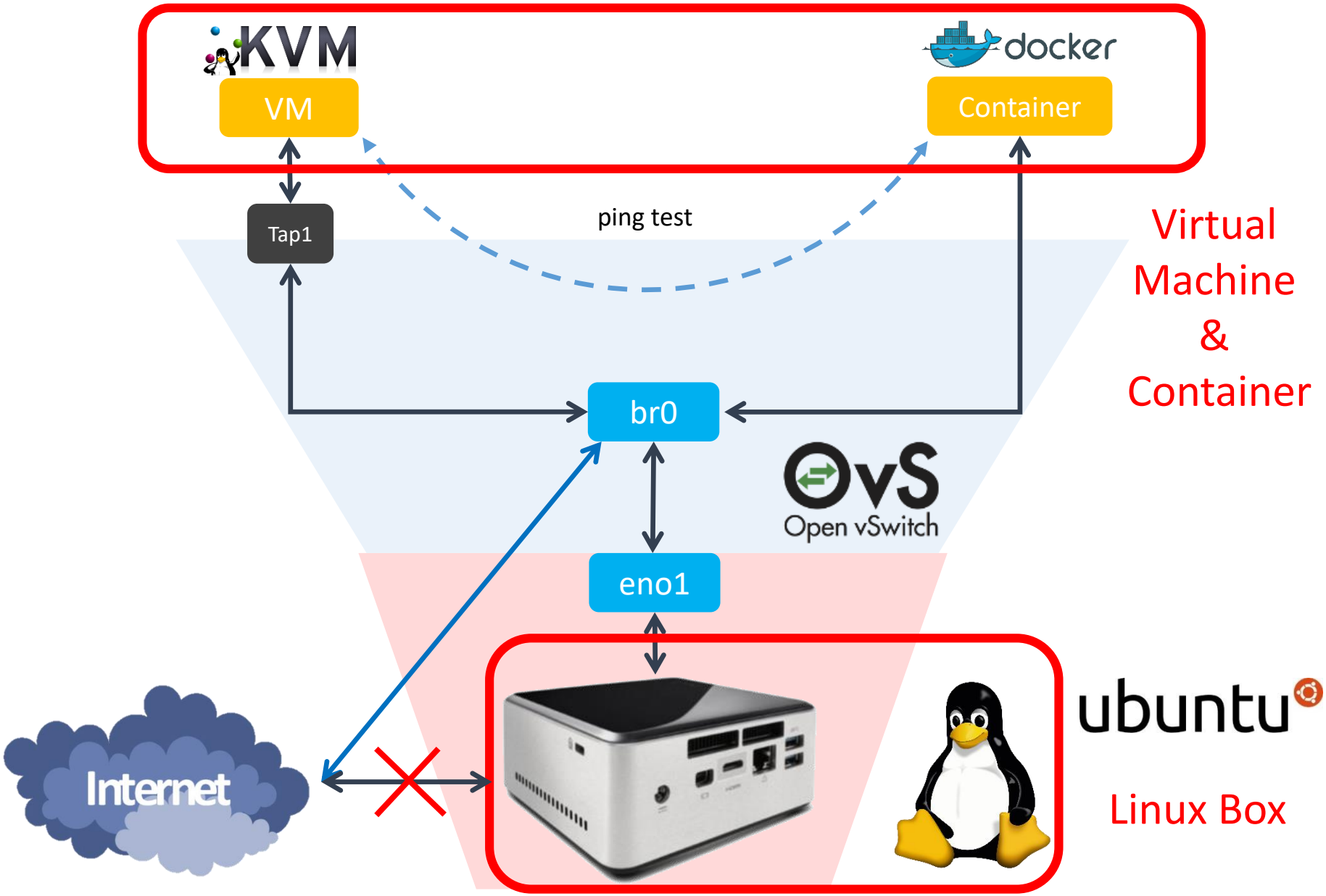
Lab Review

Lectures are divided into Lab Theory, Lab Practice and Lab Review parts.

The keyboard means that you should execute instructions by following the guidance.

Lab Theory

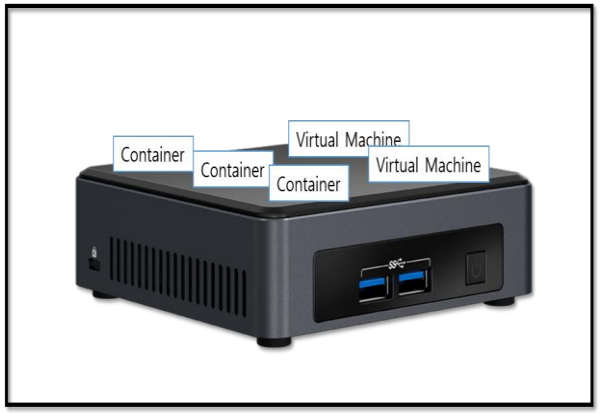
Linux Box with Virtualization/Containers



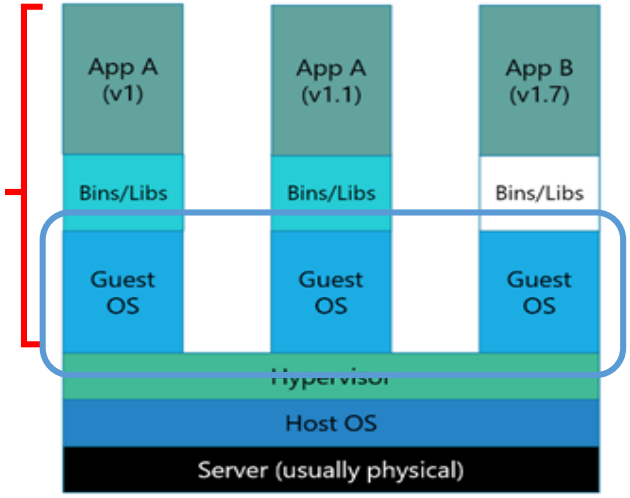
VMs and Containers on a Linux Box

On a Linux Box (e.g., NUC)

- Multiple fully-isolated (with dedicated resources), but heavy VM instances (i.e., VMs).
- Multiple partially-isolated, lightweight container instances (i.e., containers)



Virtual Machines (VMs)

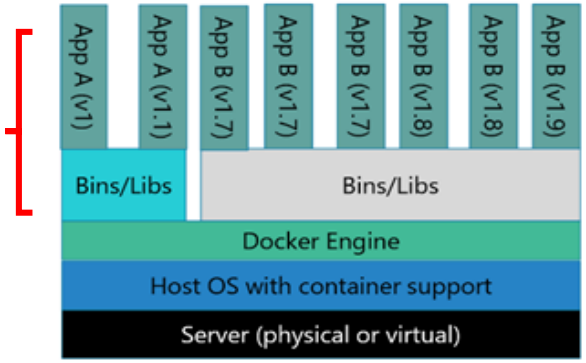


KVM

Heavier than Container!

Hypervisor Tool for creating VMs

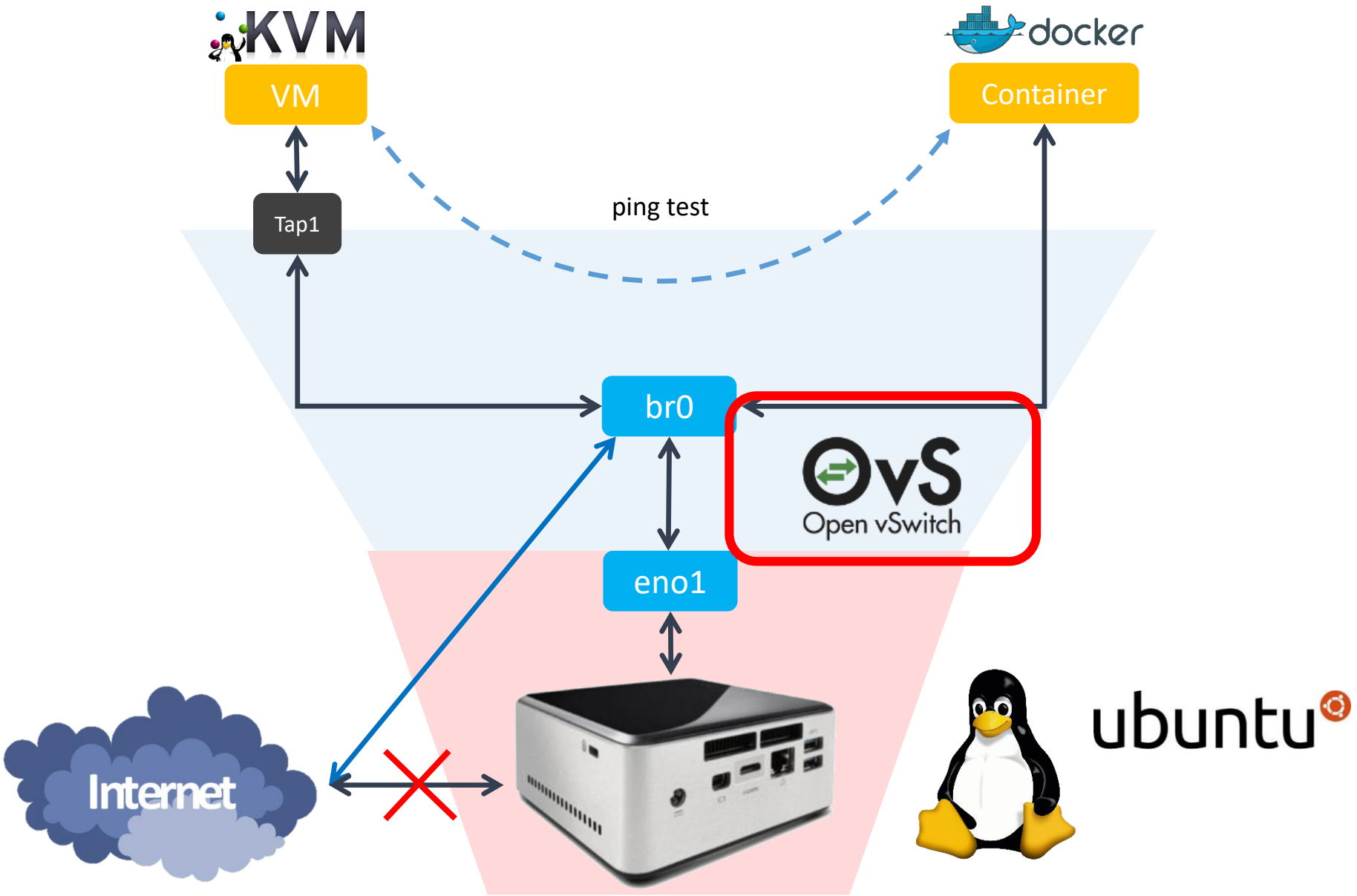
Containers



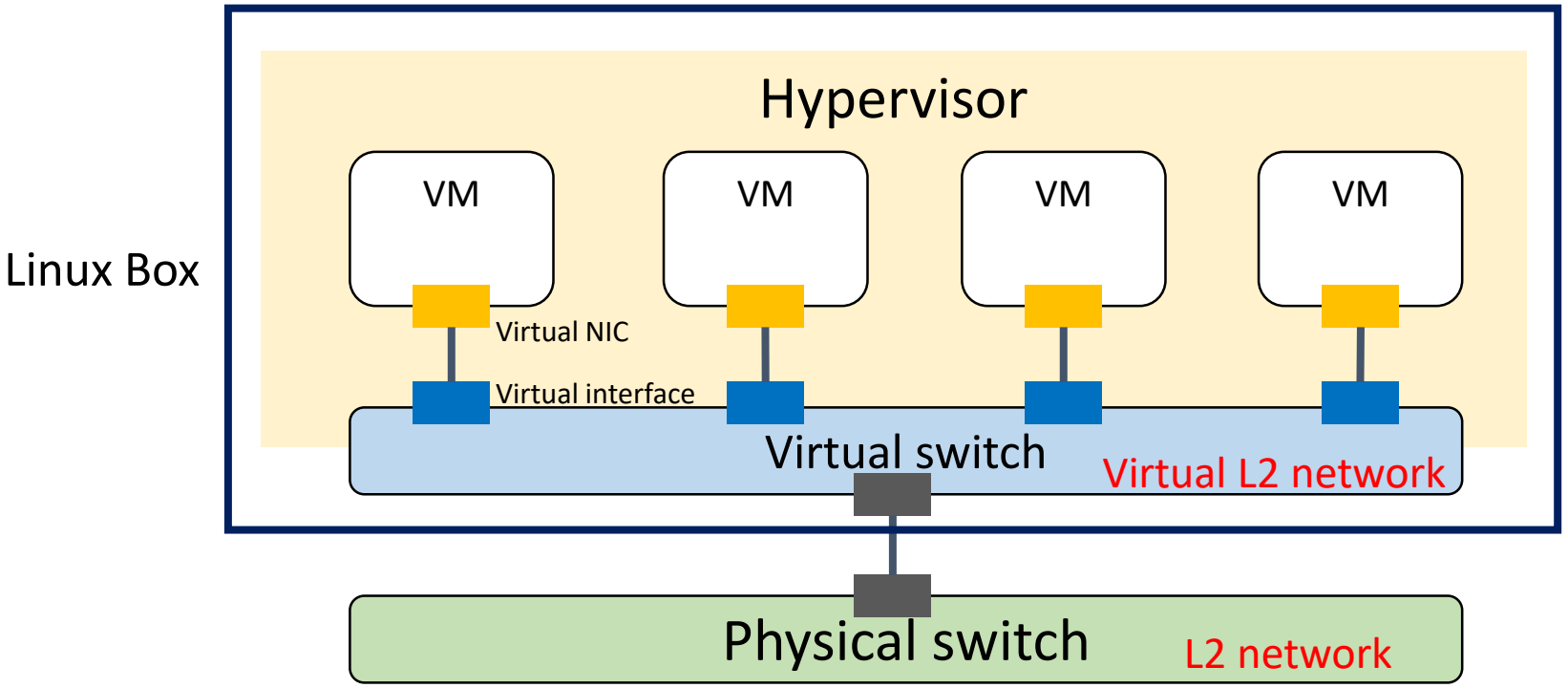
docker

Container Runtime Tool for running containers

A Switch inside Linux Box: Open vSwitch



Virtual Switch in a Box to connect VMs

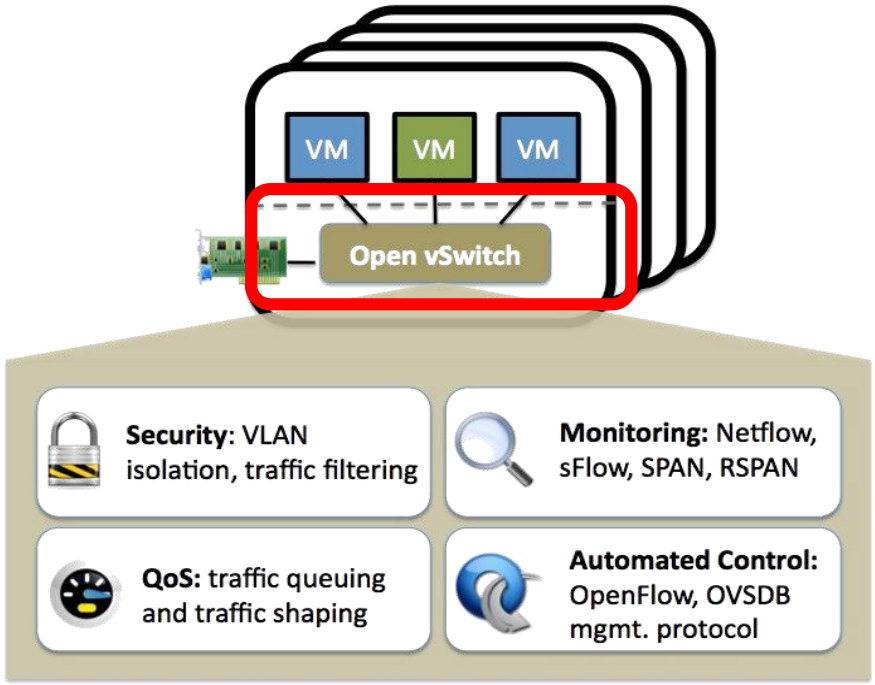


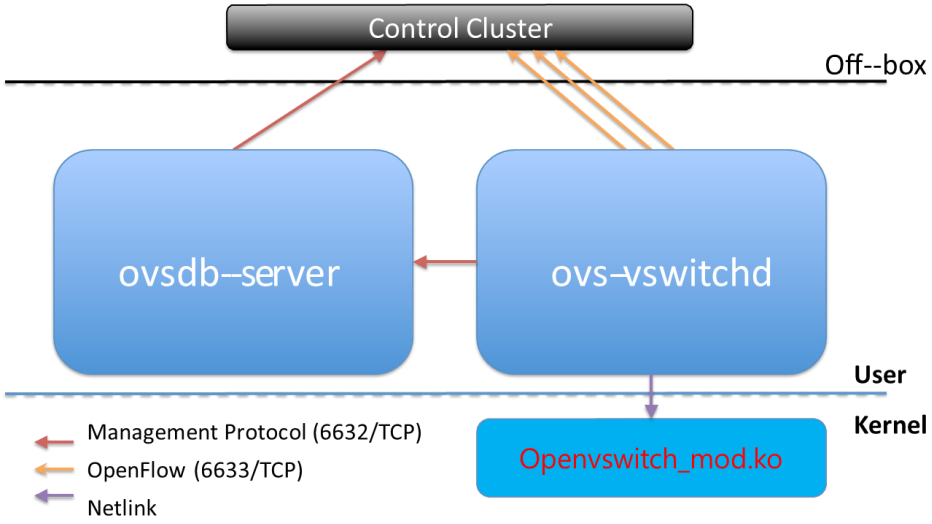
- A software-based virtual switch allows one VM to communicate with neighbor VMs as well as to connect to Internet (via physical switch).
- Software-based switches (running with the power of CPUs) are known to be more flexible/upgradable and benefited of virtualization (memory overcommit, page sharing, …)
- VMs (similarly containers) have logical (virtual) NIC with virtual Ethernet ports so that they can be plugged into the virtual interface (port) of virtual switches.

Linux-adopted virtual switch: Open vSwitch

http://openvswitch.org/

Open vSwitch is an open-source virtual switch software designed for virtual servers.





OVS Main components

Lab Practice

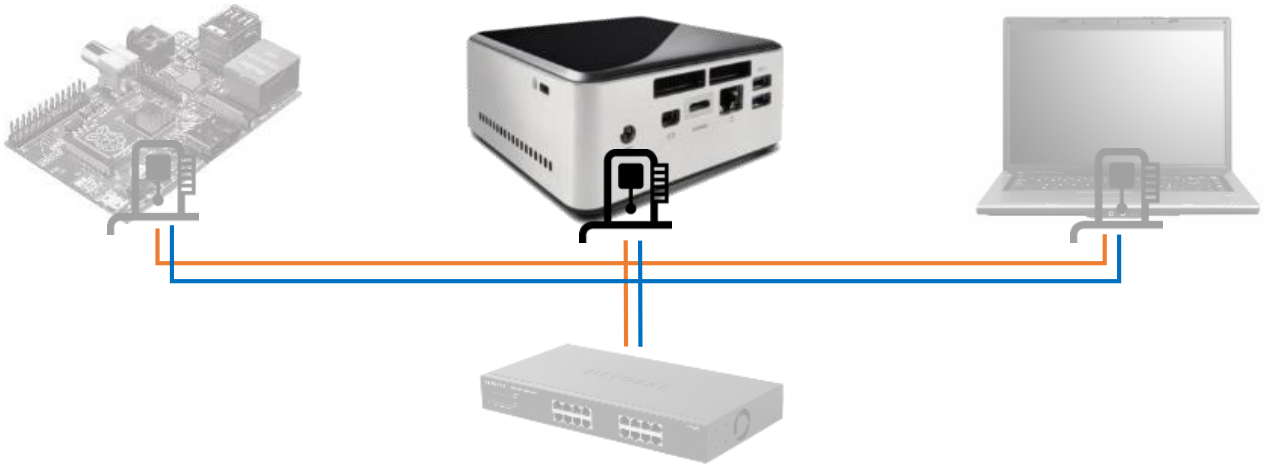
#0 - Lab Preparation

Wired connection

NAME: Raspberry Pi Model B (Pi)
CPU: ARM Cortex A7 @900MHz
CORE: 4
Memory: 1GB
SD Card: 32GB

NAME: NUC5i5MYHE (NUC PC)
CPU: Intel® Core™ i3-8109U 3.60 GHz
CORE: 2
Memory: 32GB DDR4
HDD: 94GB

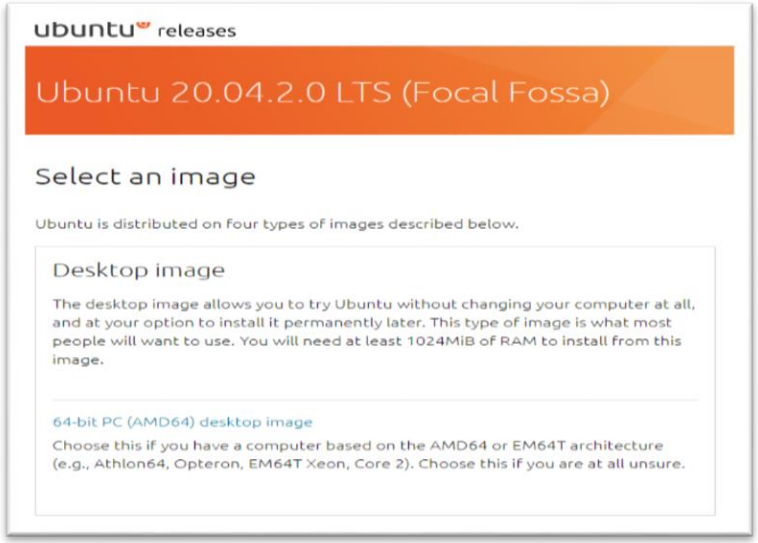
NAME: NT900X3A
CPU: i5-2537U @1.40GHz
CORE: 2
Memory: 4GB DDR3
HDD: 128GB



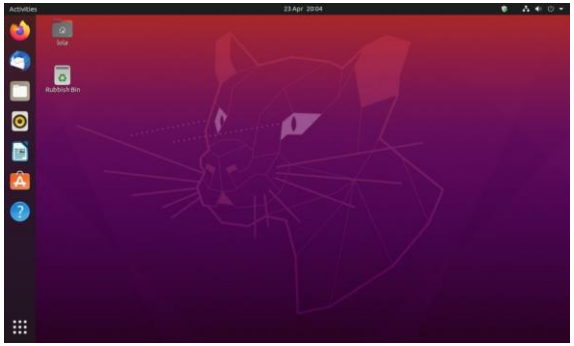
NAME: netgear prosafe 16 port gigabit switch(Switch)
Network Ports: 16 auto-sensing 10/100/1000 Mbps Ethernet ports

#1 - NUC: OS Installation

- OS : Ubuntu Desktop 20.04 LTS(64bit)
  - Download Site : https://releases.ubuntu.com/20.04/



- Installed on NUC



Ubuntu Home Screen After Installation

#1 - NUC: OS Installation

- OS Installation

Updates and other software

- Select ‘Minimal installation’

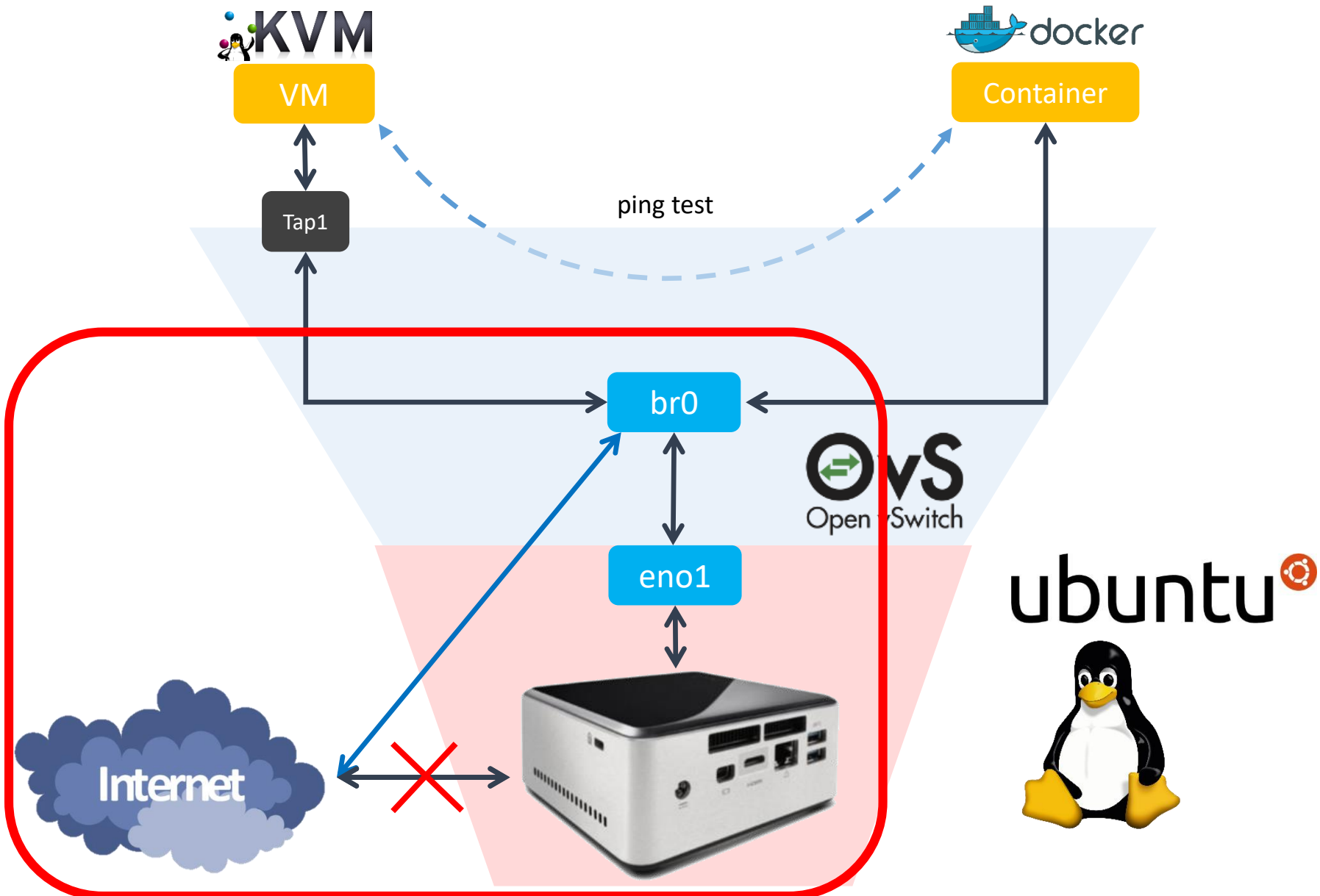
Installation type

- Select ‘Something else’
- On /dev/sda or /dev/nvme0n1
  - (UEFI), add 512MB EFI partition
  - Add empty partition with 20GB (20480MB) (Select ‘do not use the partition’)
  - Add Etc4 partition on leave memeory
- Select Boot loader
  - BIOS: Ext4 partition
  - UEFI: EFI partition

LVM관련 오류 발생 시

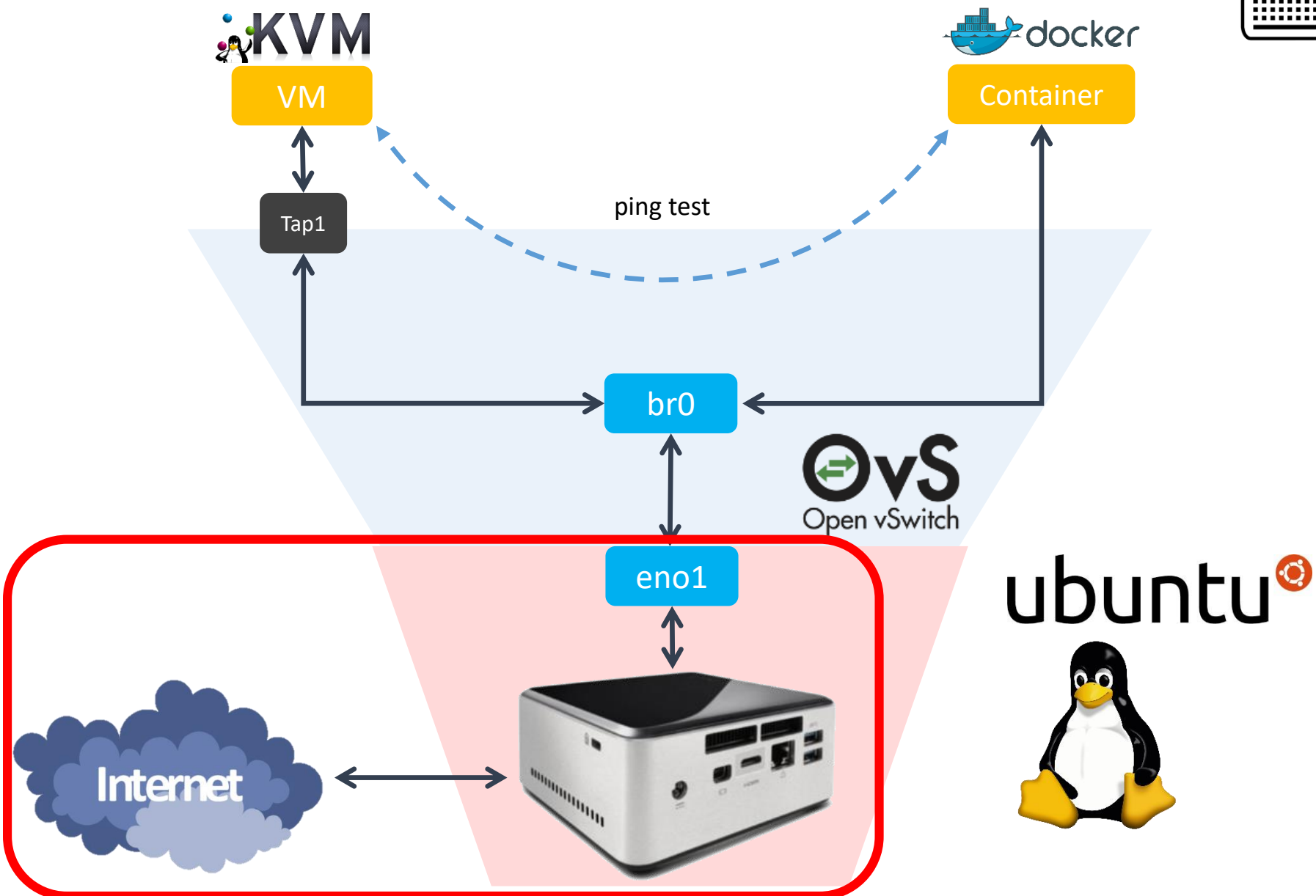
1. 뒤로 이동하여 **Installation type**화면으로 이동
2. Erase disk 선택
3. 시간대 선택화면까지 진행
4. 다시 뒤로 돌아가서 다시 **Installation type**화면으로 이동
5. Something else선택 후 정상 진행

#2 - NUC: Network Configuration



#2 - NUC: Network Configuration

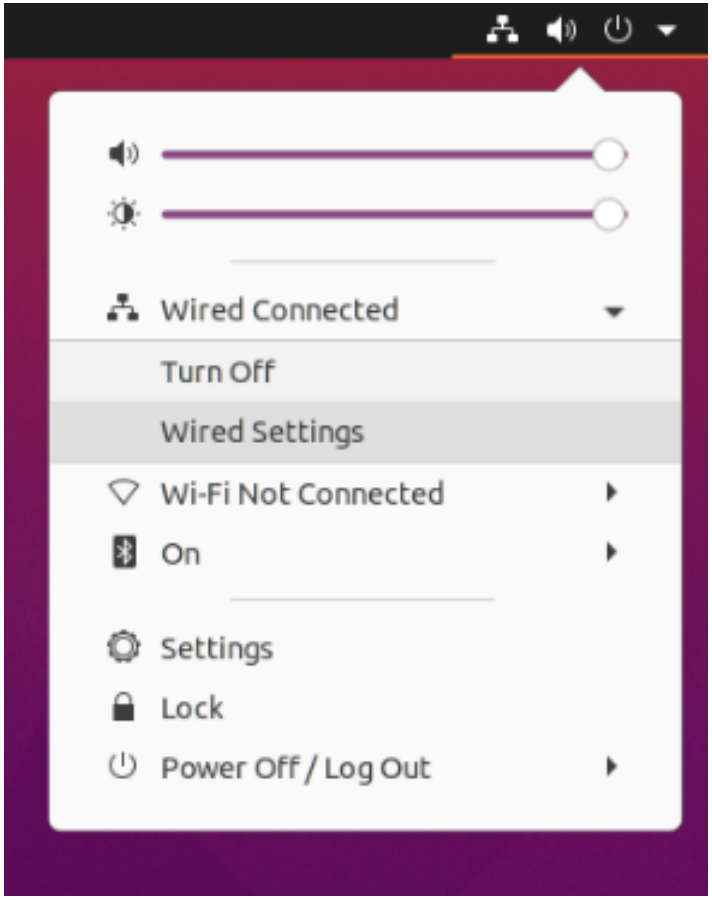
- 'Temporary' Network Configuration using GUI



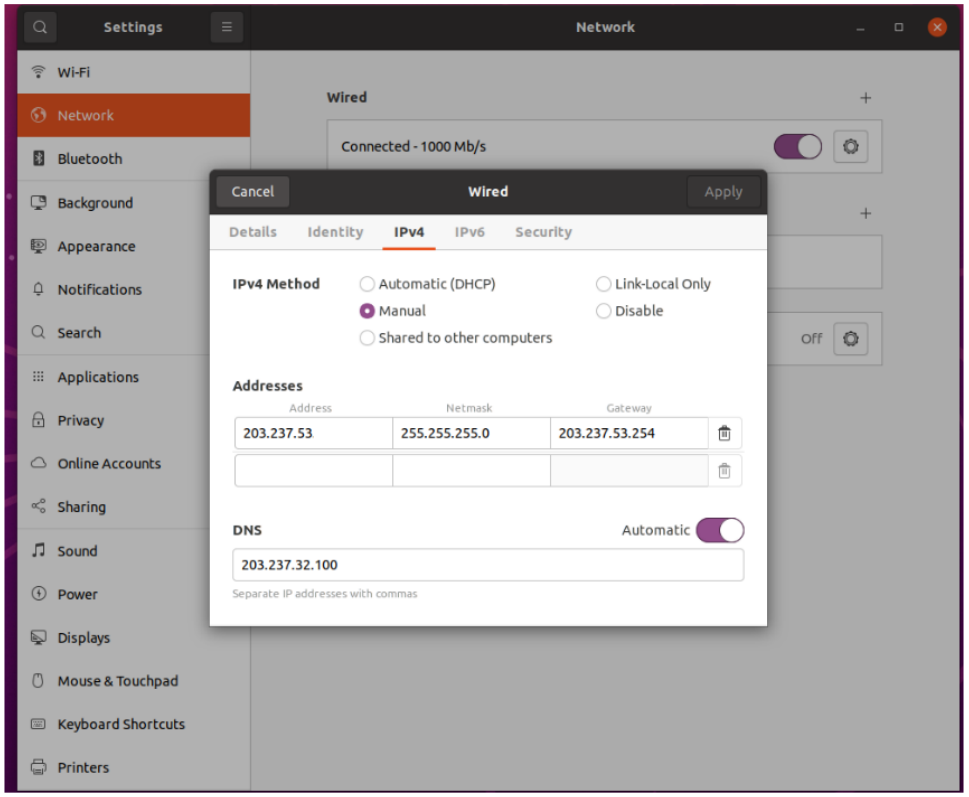
#2 - NUC: Network Configuration

- 'Temporary' Network Configuration using GUI

Click the LAN configuration icon.



Enter the network info.
(IP address, subnet mask, gateway)



#2 - NUC: Network Configuration

- Set Prerequisites

0. Update & Upgrade

```
$sudo apt update
$sudo apt upgrade
```

1. Install net-tools & ifupdown

```
$sudo apt install net-tools ifupdown
$ifconfig -a
```

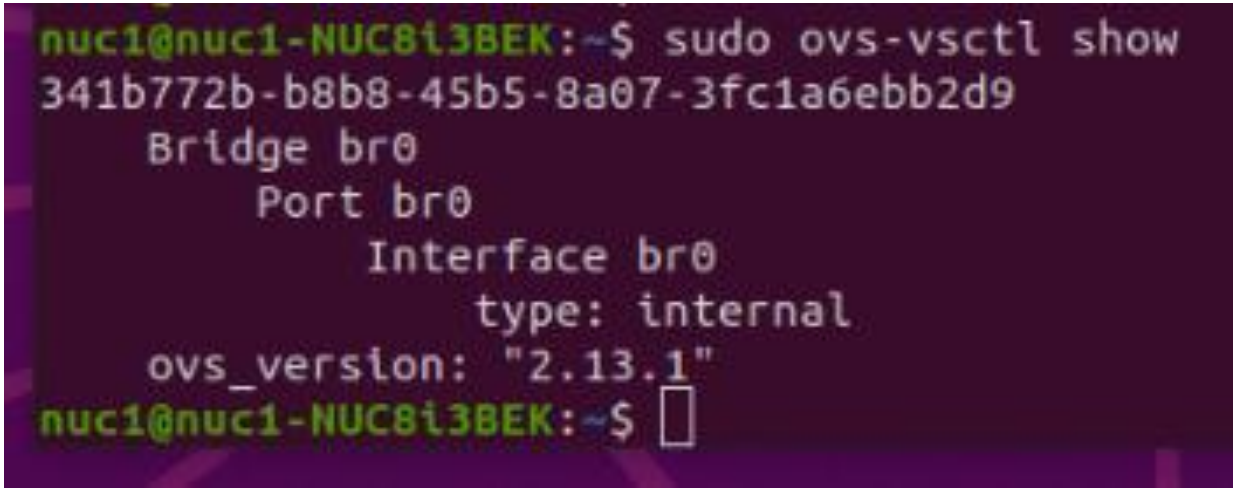
```
netcs@netcs-desktop:~$ ifconfig -a
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.199  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::82bb:956:11d6:e640  prefixlen 64  scopeid 0x20<link>
        ether ec:a8:6b:fb:a2:09  txqueuelen 1000  (Ethernet)
        RX packets 255746  bytes 382878532 (382.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40500  bytes 3146671 (3.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xf7c00000-f7c20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2771  bytes 237705 (237.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2771  bytes 237705 (237.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

#2 - NUC: Network Configuration

- Set Prerequisites

2. Install openvswitch-switch & make br0 bridge

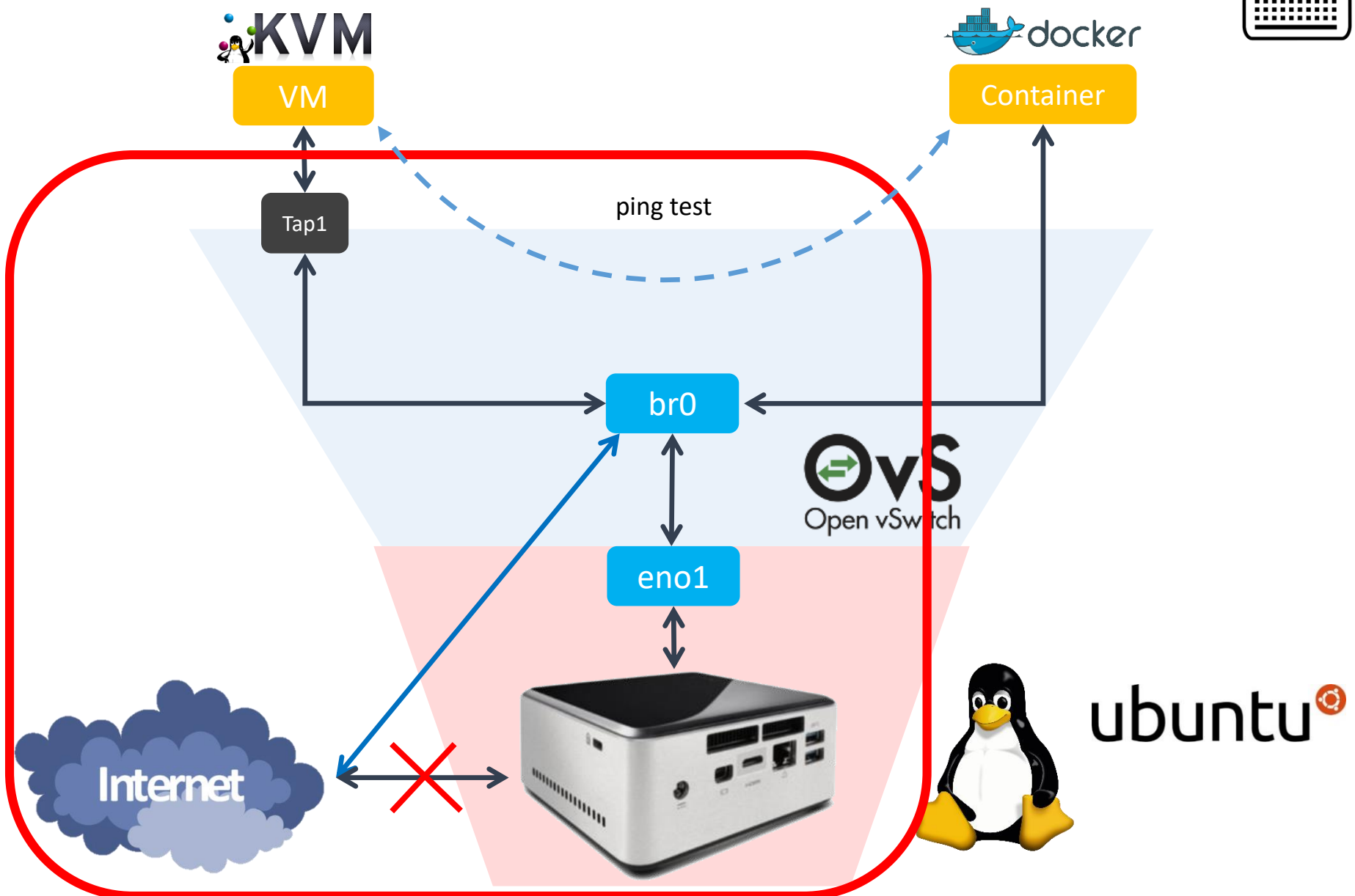
```
$sudo apt intall openvswitch-switch
$sudo ovs-vsctl add-br br0
$sudo ovs-vsctl show
```



ovs-vsctl show

#2 - NUC: Network Configuration

- Network interface

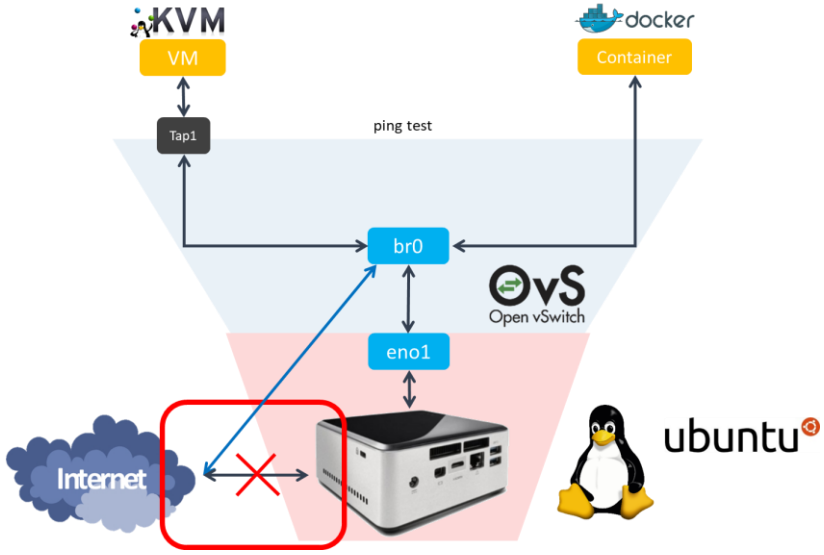


#2 - NUC: Network Configuration

- Network interface

Disable netplan

$sudo su
#systemctl stop systemd-networkd.socket systemd-networkd networkd-dispatcher systemd-networkd-wait-online
#systemctl disable systemd-networkd.socket systemd-networkd networkd-dispatcher systemd-networkd-wait-online
#systemctl mask systemd-networkd.socket systemd-networkd networkd-dispatcher systemd-networkd-wait-online
#apt-get --assume-yes purge nplan netplan.io
#exit



#2 - NUC: Network Configuration

- Network interface

Open /etc/network/interfaces via 'vi'

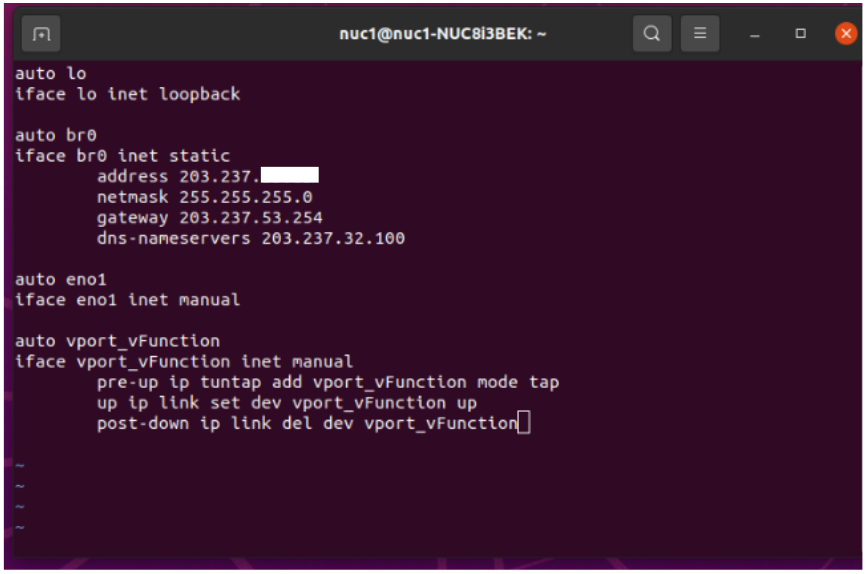
$sudo vi /etc/network/interfaces

Configurate the Network Interface vport_vFunction is a tap interface and attach it to your VM.

---------- /etc/network/interfaces ----------

```
auto lo
iface lo inet loopback

auto br0
iface  br0 inet static
        address ---your nuc ip---
        netmask 255.255.255.0
        gateway ---gateway ip---
        dns-nameservers 8.8.8.8

auto eno1
iface eno1 inet manual

auto vport_vFunction
iface vport_vFunction inet manual
        pre-up ip tuntap add vport_vFunction mode tap
        up ip link set dev vport_vFunction up
        post-down ip link del dev vport_vFunction
```

Some NUC have different Interface name. So you need to check your NUC's interface name using 'ip a' command.

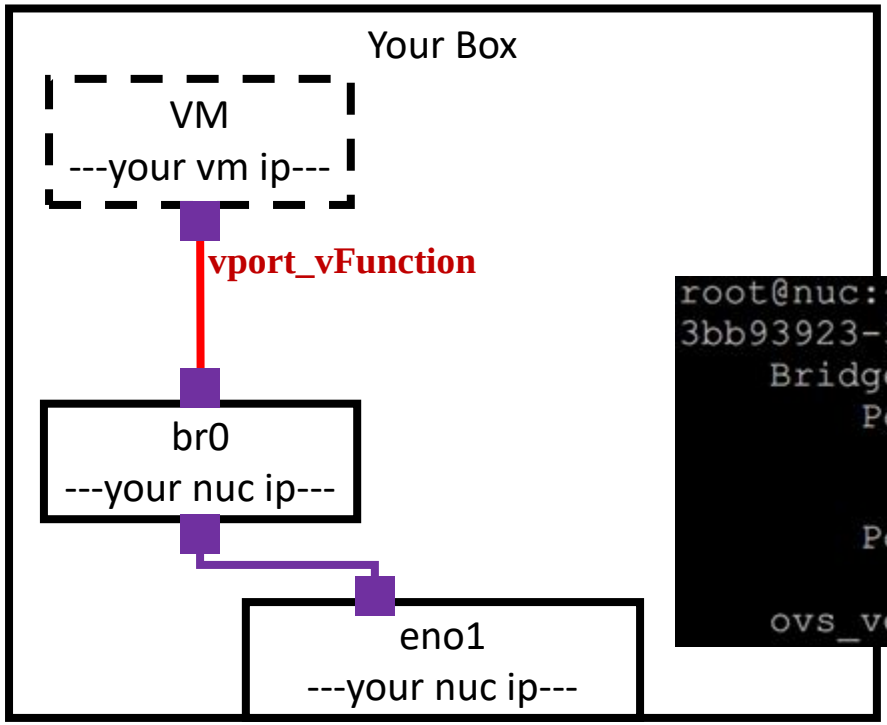


#2 - NUC: Network Configuration

- Network interface

We will make VM attaching vport_vFunction. You can think this tap as a NIC of VM.

Below is the figure you configurated so far




```
root@nuc:~# ovs-vsctl show
3bb93923-3eac-420a-9da9-9143aff14209
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port vport_vFunction  → [tap_name]
            Interface vport_vFunction
    ovs_version: "2.0.2"
```

#2 - NUC: Network Configuration

- Network interface

Restrart the whole interfaces 1

```
$sudo su
#systemctl unmask networking
#systemctl enable networking
#systemctl restart networking
```

add port 'eno1' and 'vport_vFunction' to 'br0'

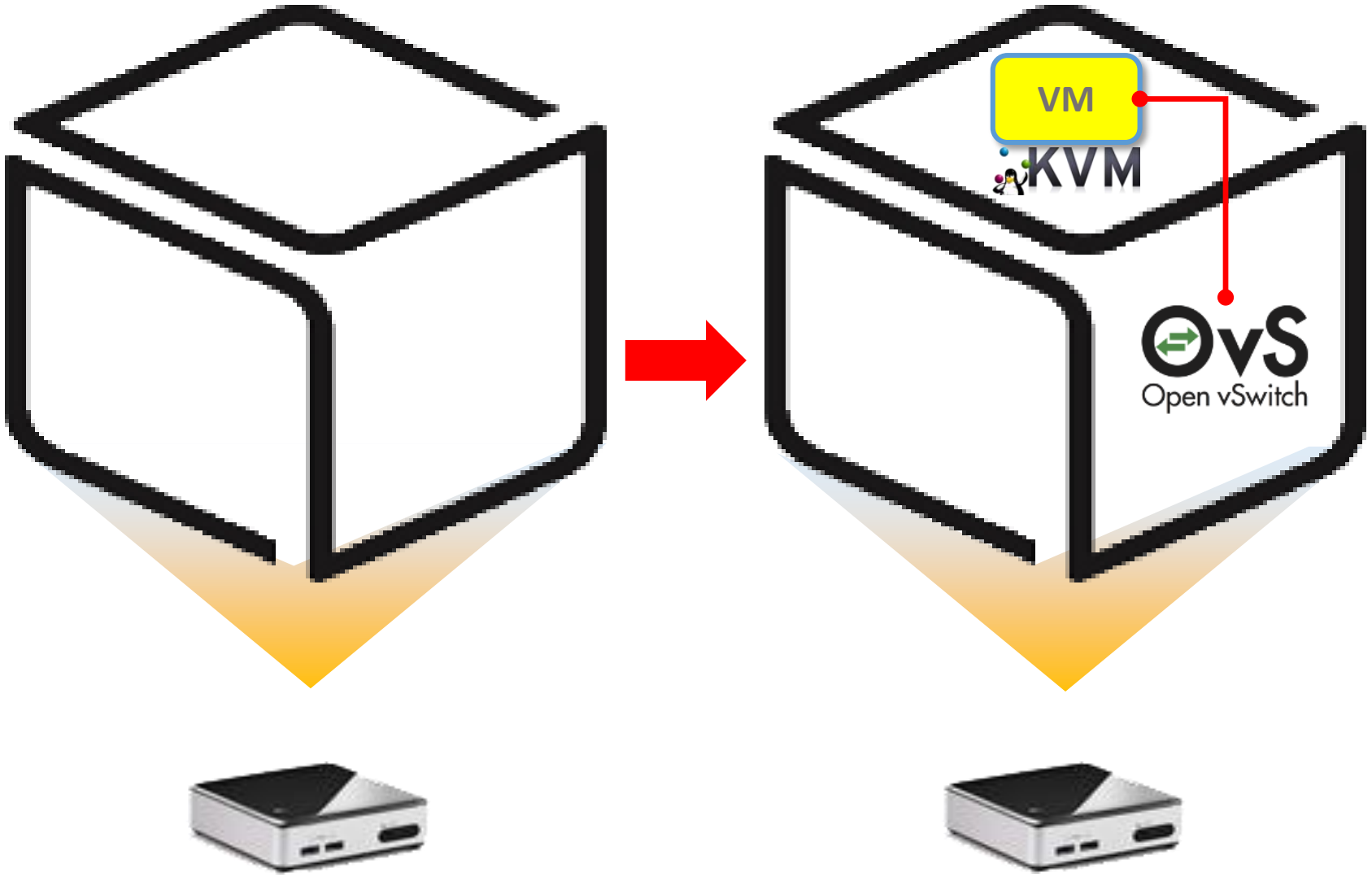
```
$ sudo ovs-vsctl add-port br0 eno1
$ sudo ovs-vsctl add-port br0 vport_vFunction
$ sudo ovs-vsctl show
```

Restrart the whole interfaces 2

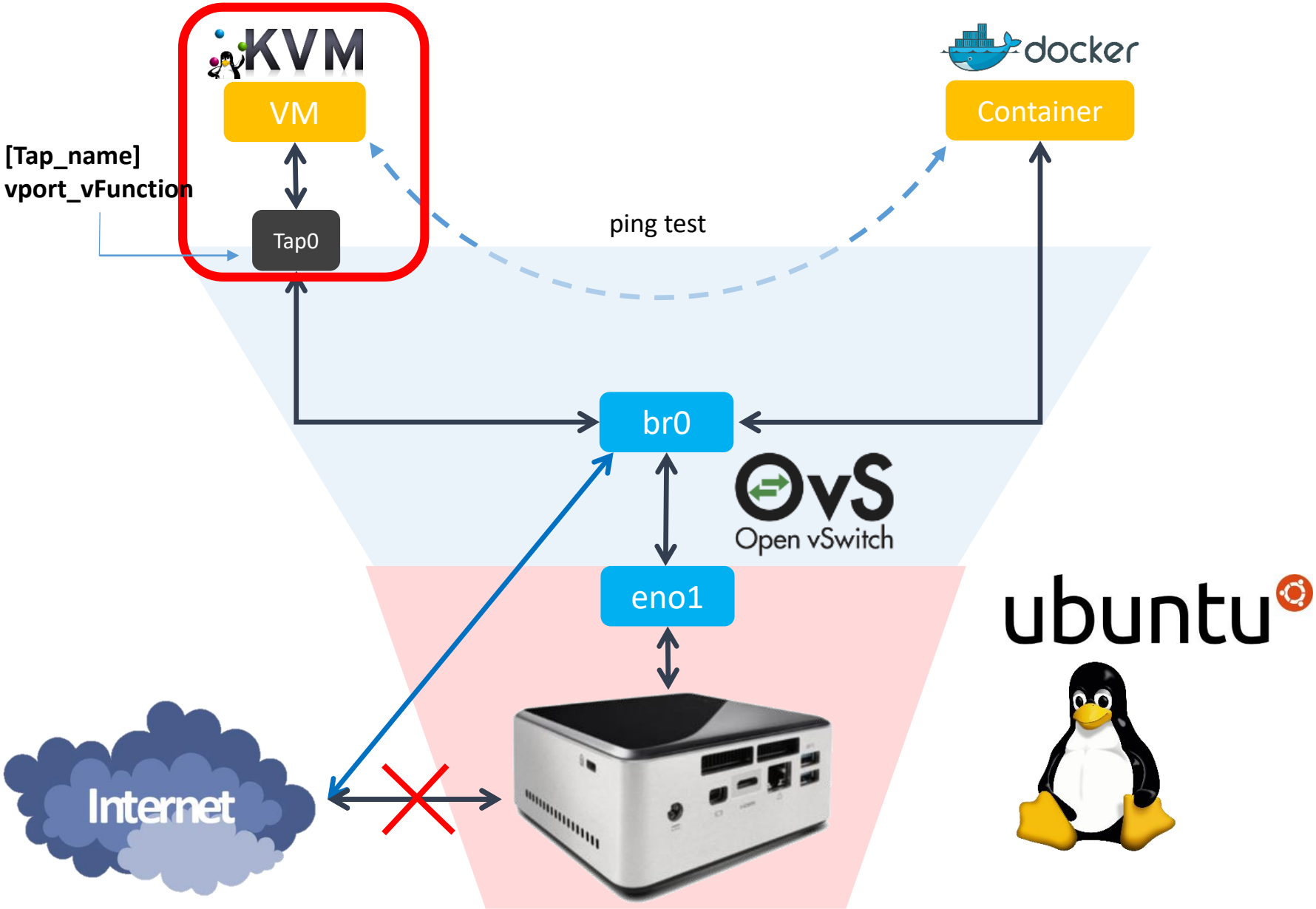
```
#systemctl unmask networking
#systemctl enable networking
#systemctl restart networking
#exit
```

KVM-based VM connected via OVS

- Goal of this section



#3 - NUC: Making VM with KVM



#3 - NUC: Making VM with KVM

-Install dependency to upgrade KVM

Install dependency & download Ubuntu 20.04 64bit server image.

```
$sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils
//upgrade KVM
//qemu is open-source emulator

$wget http://old-releases.ubuntu.com/releases/focal/ubuntu-20.04-beta-live-server-amd64.iso
```

Now we are ready to make VM. So continue the setting.

#3 - NUC: Making VM with KVM

-Prepare for Ubuntu VM

Make a VM image.

```
$sudo qemu-img create [img_name].img -f qcow2 [storage_capacity]
```

```
$sudo qemu-img create vFunction20.img -f qcow2 10G
```

Boot VM image from Ubuntu iso file (**띄어쓰기 주의!**).

```
$sudo kvm -m [memory_capacity] -name [vm_name] -smp cpus=[#cpu],maxcpus= [#maxcpu] -device virtio-net-pci,netdev=net0 -netdev tap,id=net0,ifname= [tap_name],script=no -boot d [img_name].img -cdrom ubuntu-20.04-beta-live-server-amd64.iso -vnc :[#] –daemonize -monitor telnet:127.0.0.1:3010,server,nowait,ipv4
```

```
$ sudo kvm -m 1024 -name tt -smp cpus=2,maxcpus=2 -device virtio-net-pci,netdev=net0 -netdev tap,id=net0,ifname=vport_vFunction,script=no -boot d vFunction20.img -cdrom ubuntu-20.04-beta-live-server-amd64.iso -vnc :5 -daemonize -monitor telnet:127.0.0.1:3010,server,nowait,ipv4
```

#3 - NUC: Making VM with KVM

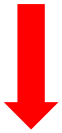
-Prepare for Ubuntu VM

Configure SNAT with iptables for VM network

```
$sudo iptables –A FORWARD –i eno1 –j ACCEPT
$sudo iptables –A FORWARD –o eno1 –j ACCEPT
$sudo iptables –t nat –A POSTROUTING –s 192.168.100.0/24 –o eno1 –j SNAT --to <Your ip address>
```

$vi /etc/sysctl.conf

#net.ipv4.ip_forward=1



net.ipv4.ip_forward=1

$sysctl –p

net.ipv4.ip_forward = 1

Configuration complete

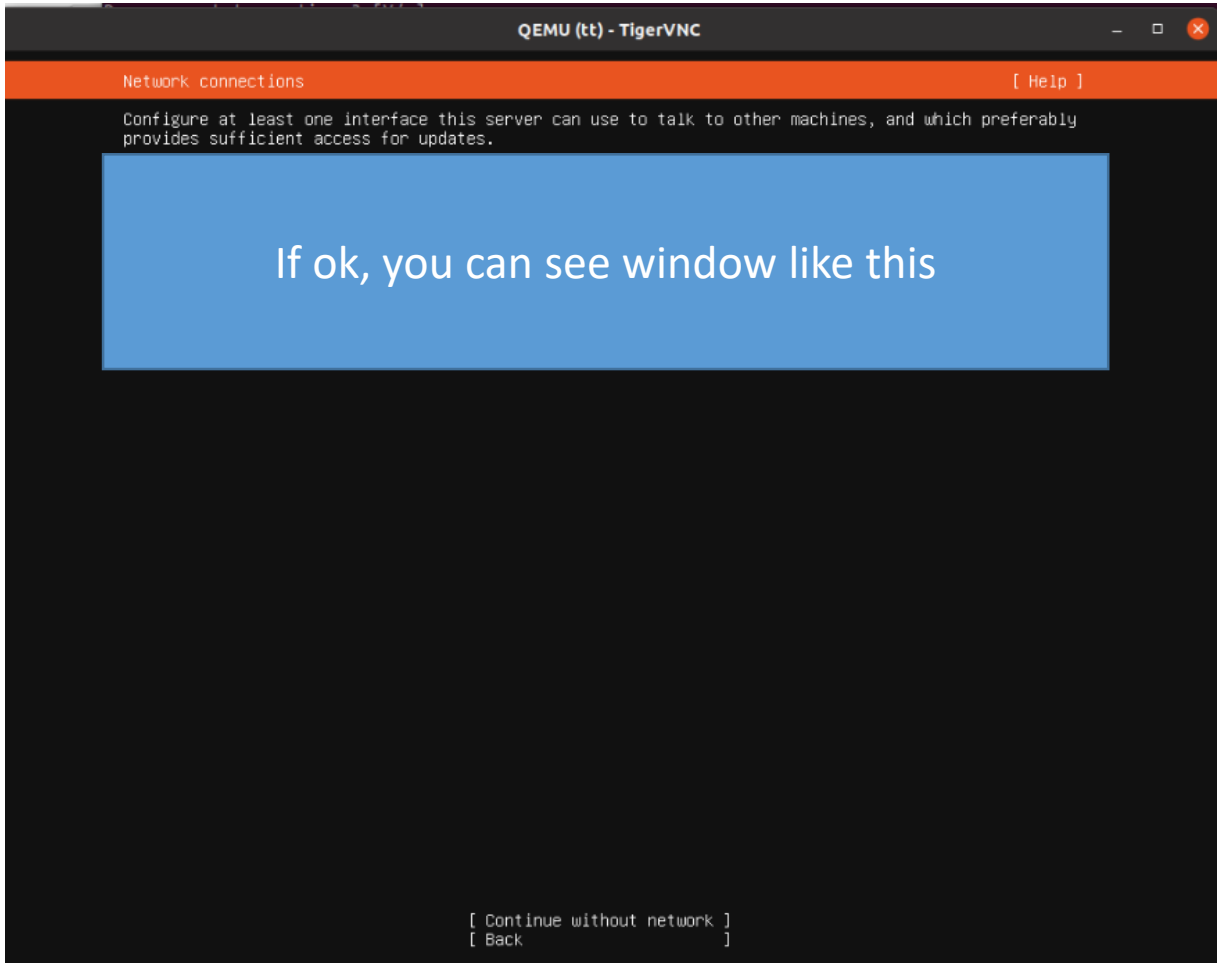
```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1
```

#3 - NUC: Making VM with KVM

-Install Ubuntu VM (control with 'Enter key' and 'Arrow keys')

Install VNC viewer and see inside of VM

```
$ sudo apt-get install tigervnc-viewer
$ vncviewer localhost :5
```



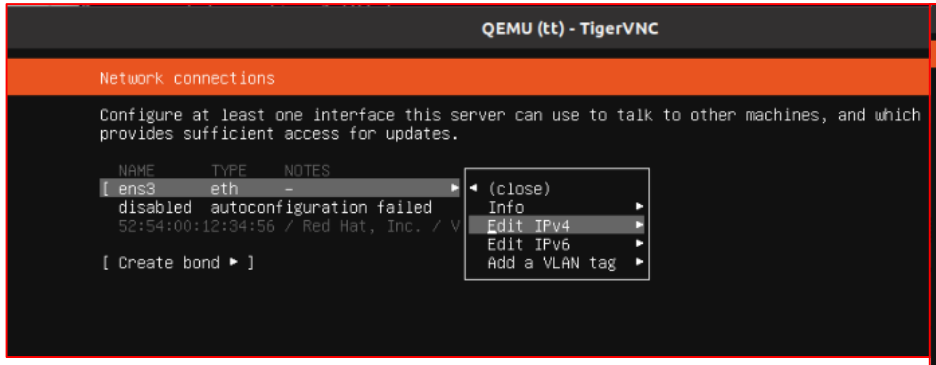
#3 - NUC: Making VM with KVM

- VM network configuration (control with 'Enter key' and 'Arrow keys')

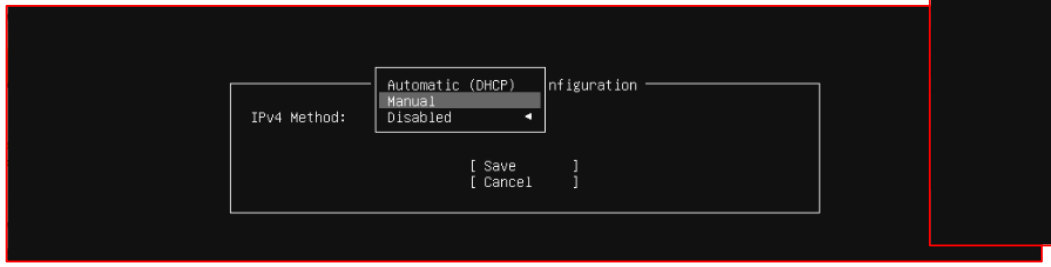
1. 네트워크 디바이스 선택 -> Edit IPv4
2. IPv4 Method -> Manual
3. 설정

   subnet: 203.237.53.0/24
   Address: **---your vm ip---**
   Gateway: **---gateway ip---**
   Name Servers: 8.8.8.8

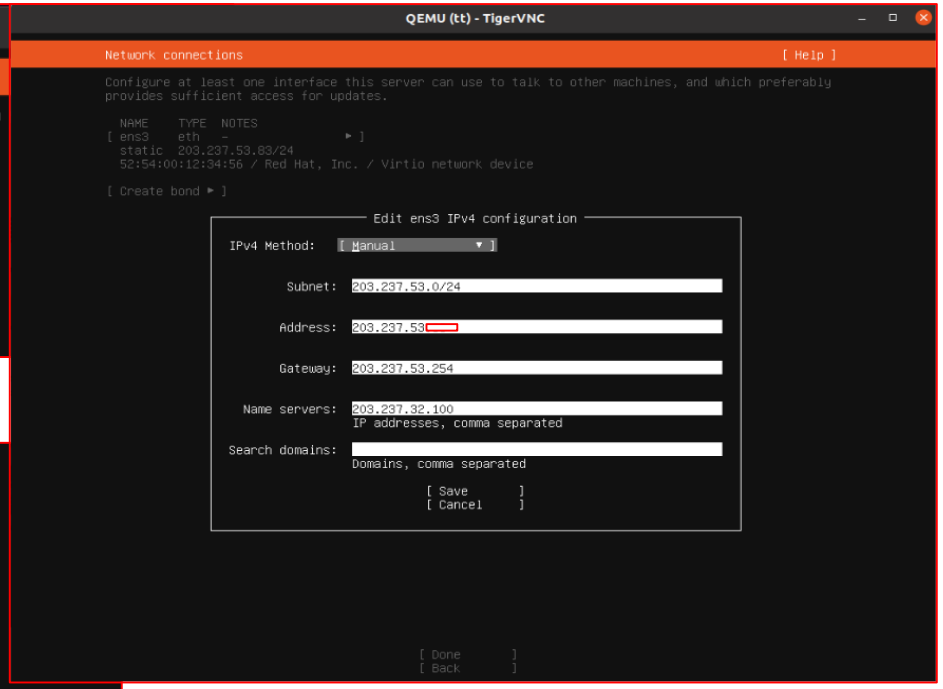
1. Select network device 'Edit IPv4'



2. Select 'Manual'



3. Enter the network info.



#3 - NUC: Making VM with KVM

- Installation Completed (control with 'Enter key' and 'Arrow keys')

When 'installation completed' message is shown, terminate the VM

```
$ sudo killall -9 qemu-system-x86_64
```

boot VM again (mac should be different from others).

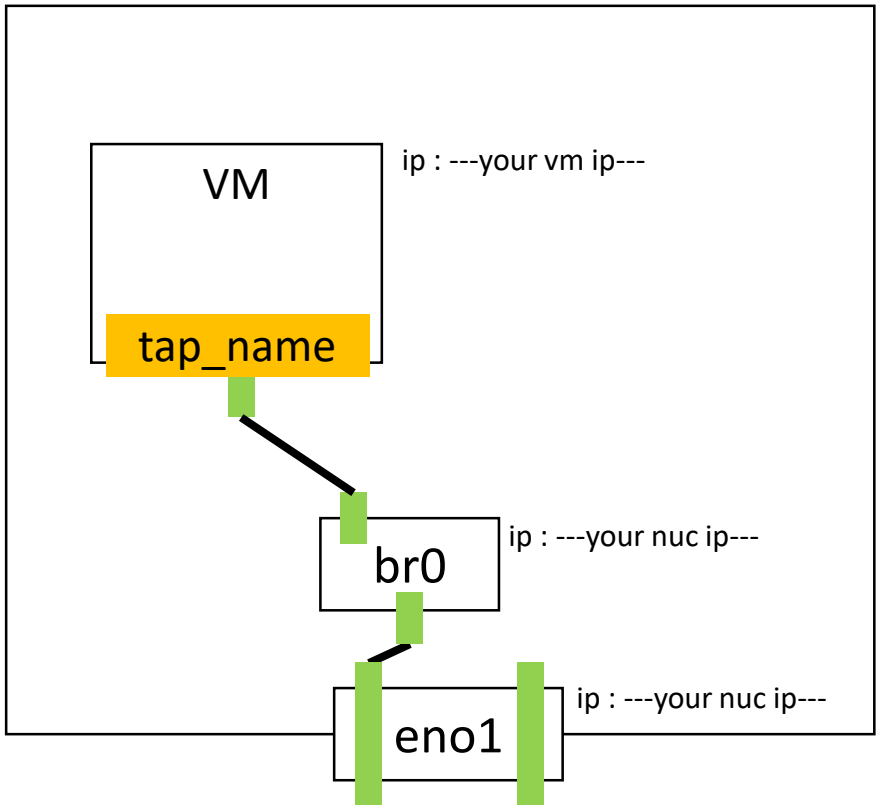
```
$sudo kvm -m [memory capacity] -name [name] -smp cpus=[#cpu],maxcpus= [#maxcpu] -device virtio-net-pci,netdev=net0 –netdev tap,id=net0,ifname= [tap_name],script=no -boot d [name].img -vnc : [#] -daemonize
```

```
$ sudo kvm -m 1024 -name tt -smp cpus=2,maxcpus=2 -device virtio-net-pci,netdev=net0 -netdev tap,id=net0,ifname=vport_vFunction,script=no -boot d vFunction20.img -vnc :5 -daemonize
```

#5 - OVS connects with KVM

- Check situation

```
root@nuc:~# ovs-vsctl show
3bb93923-3eac-420a-9da9-9143aff14209
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port vport_vFunction
            Interface vport_vFunction
    ovs_version: "2.0.2"
```



#6 - NUC: Installing ssh in VM

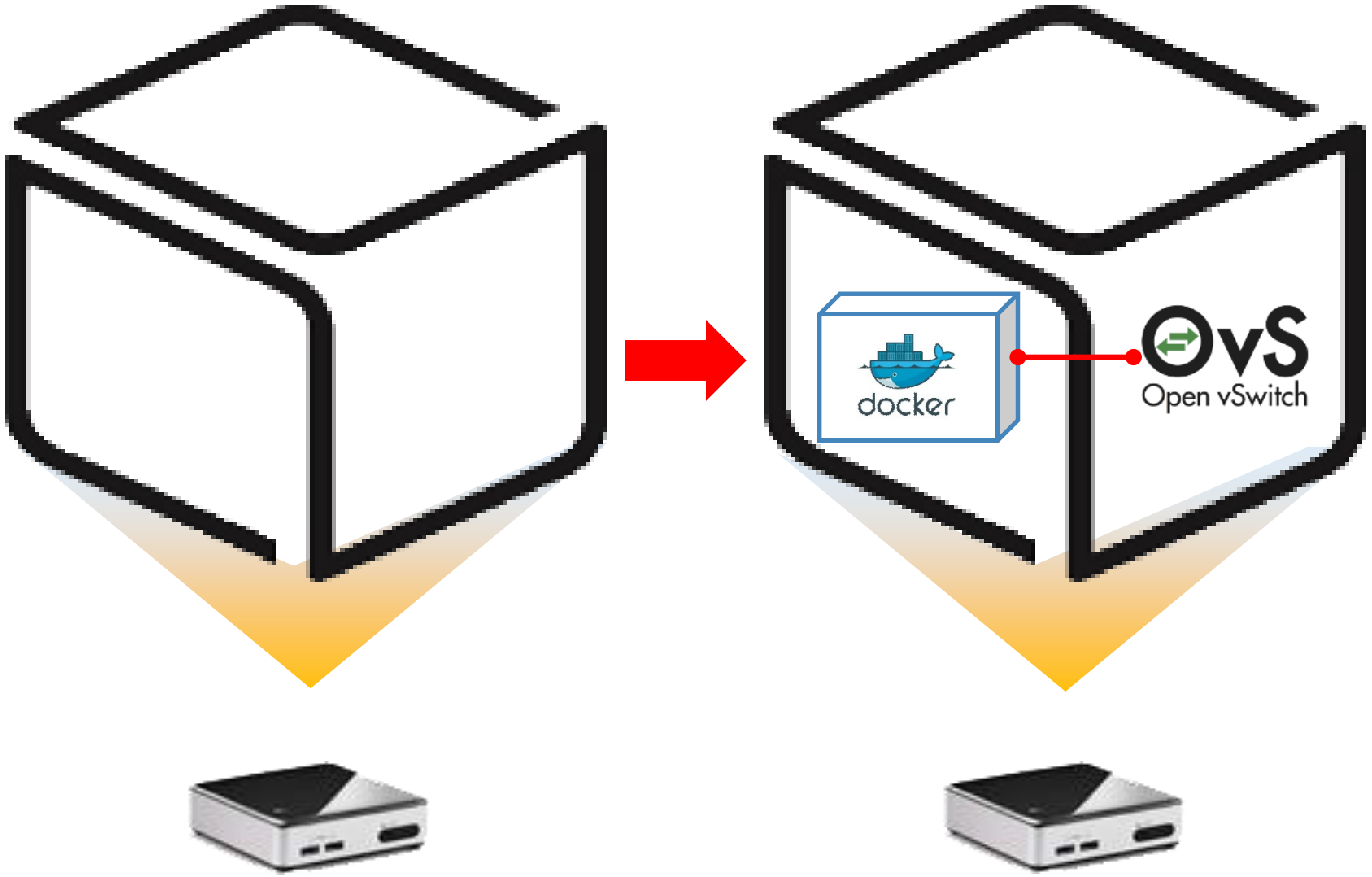
- Don't forget to install ssh in VM

In VMs,

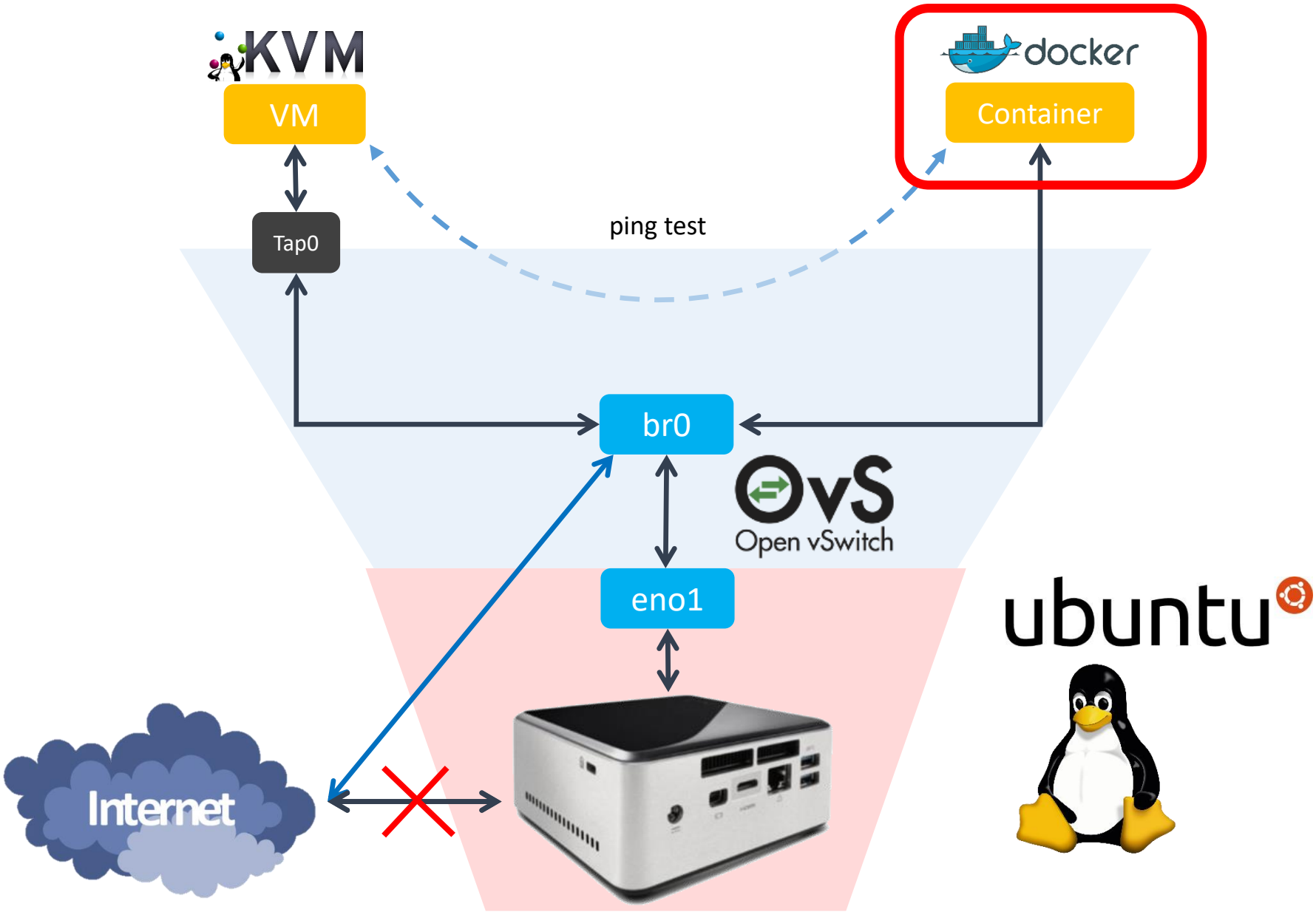
```
$sudo apt-get update
$sudo apt-get install net-tools ssh -y
```

```
nuc@nuc:~$ ssh vbox@192.168.0.3
The authenticity of host '192.168.0.3 (192.168.0.3)' can't be established.
ECDSA key fingerprint is da:c5:2c:53:5a:6f:b4:3c:03:02:04:f3:6a:17:ca:ab.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.3' (ECDSA) to the list of known hosts.
vbox@192.168.0.3's password:
```

Docker Container connected via OVS
- Goal of this section



#7 - Making a Docker Container



#7 - Making a Docker Container

- Docker installation

In NUC,

Docker installation.

```
// Set up the repository:
// Install packages to allow apt to use a repository over HTTPS
$ sudo apt-get update && sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg2

// Add Docker's official GPG key:
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key - -keyring /etc/apt/trusted.gpg.d/docker.gpg add –

// Add the Docker apt repository:
$ sudo add-apt-repository \
$              "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$               $(lsb_release -cs) \
$              stable"

// Install Docker CE
$ sudo apt-get update && sudo apt-get install -y \
$              containerd.io=1.2.13-2 \
$              docker-ce=5:19.03.11~3-0~ubuntu-$(lsb_release -cs) \
$              docker-ce-cli=5:19.03.11~3-0~ubuntu-$(lsb_release -cs)

// Create /etc/docker
$ sudo mkdir -p /etc/docker
```

cont‘d on next page

#7 - Making a Docker Container

- Docker installation

In NUC,

Docker installation (cont'd)

```
//continued
// Set up the Docker daemon
$ cat <<EOF | sudo tee /etc/docker/daemon.json
$ {
$           "exec-opts": ["native.cgroupdriver=systemd"],
$           "log-driver": "json-file",
$           "log-opts": {
$                       "max-size": "100m"
$           },
$           "storage-driver": "overlay2"
$ }
$  EOF

// Create /etc/systemd/system/docker.service.d
$ sudo mkdir -p /etc/systemd/system/docker.service.d

$ sudo systemctl daemon-reload
$ sudo systemctl enable docker
$ sudo systemctl restart docker
$ sudo systemctl restart docker.socket
```

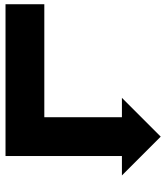
#7 - Making a Docker Container

- Docker installation

In NUC,

Now Installation completed. Let's Check it

```
$ sudo docker run hello-world
```



```
Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/userguide/
```

reference: http://docs.docker.com/linux/step_one/

#7 - Making a Docker Container

- Make container

Run docker container. (**Remember the container_name)

$sudo docker run -it --net=none --name [container_name] ubuntu /bin/bash

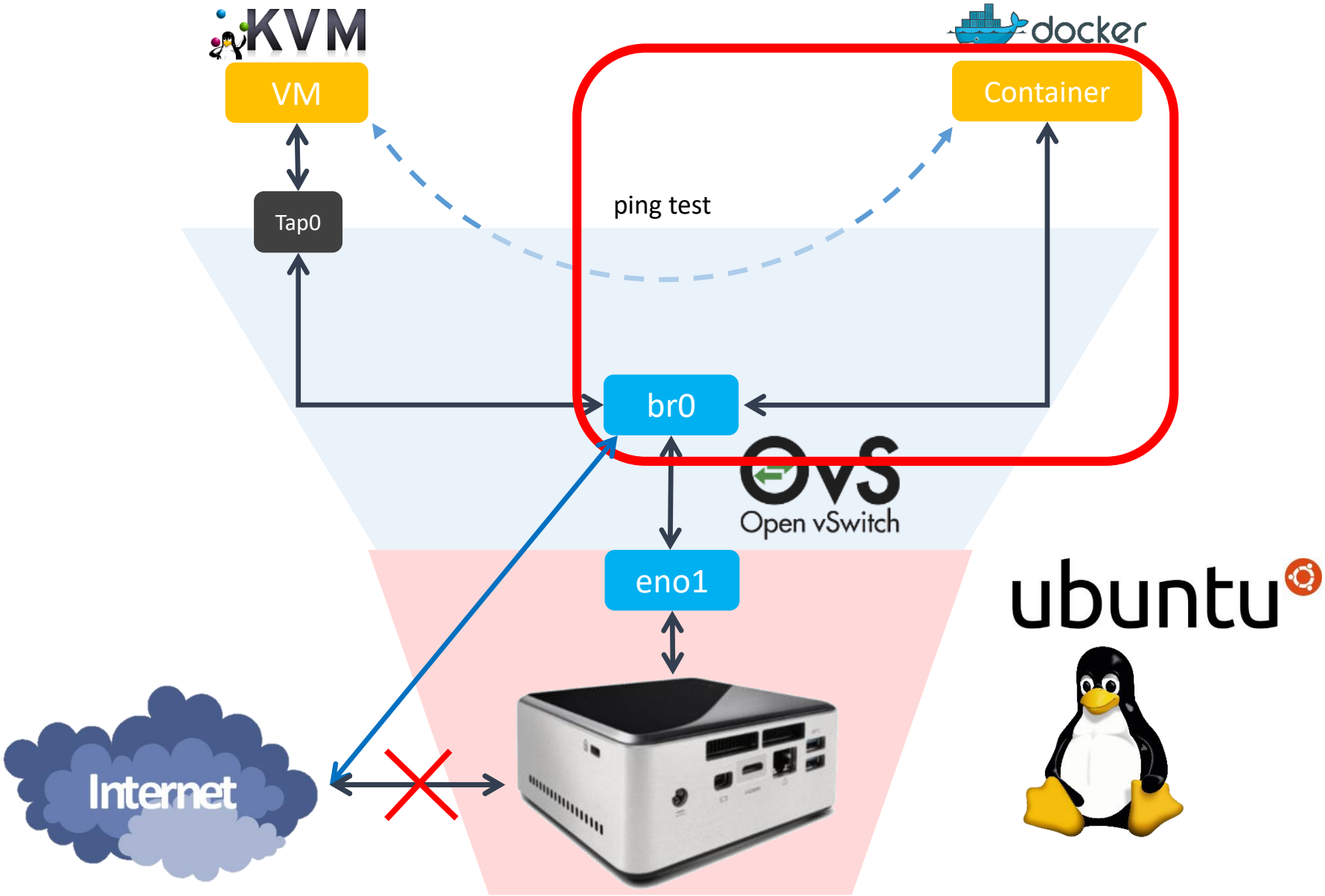
```
nuc@nuc:~$ docker run -it --net=none --name c1 ubuntu /bin/bash
root@8346684676d8:/#
```

※ctrl + p, q → detach docker container

※docker attach [container_name] → get into docker container console

#8 - Connect Docker Container

- Connect with OVS bridge



#8 - Connect docker Container

- Connect with OVS bridge

Install OVS-docker utility in host machine. (Not in inside of Docker container.)

```
$sudo ovs-docker add-port br0 veno1 [container_name] --ipaddress=---your docker ip---/24 --gateway=---gateway ip---


$sudo docker attach [container_name] //enter to docker container
#apt-get update
#apt-get install net-tools
#apt-get install iputils-ping
```

#9 – Keep Docker network configuration

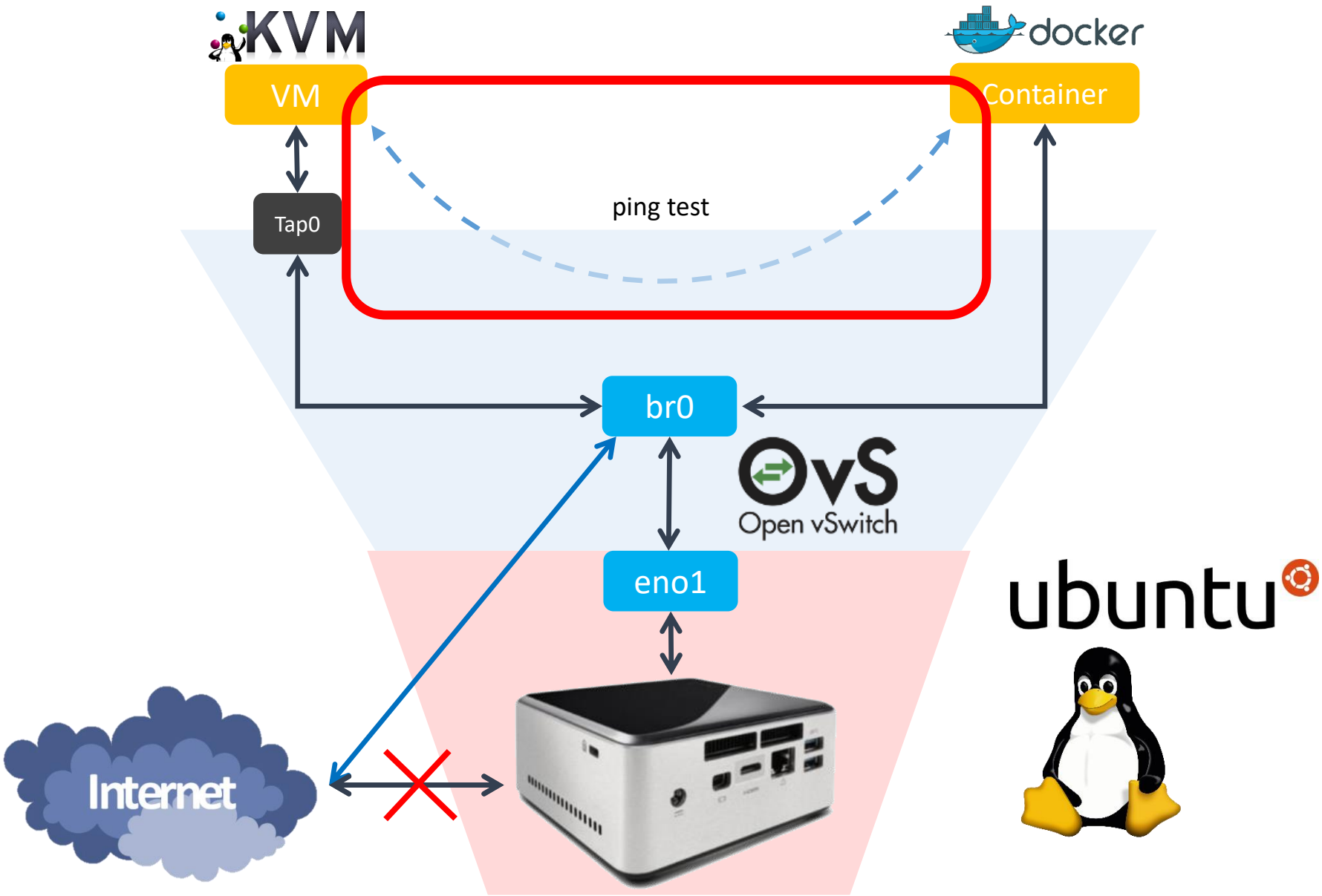
- /etc/rc.local

Modify /etc/rc.local

```
$sudo vi /etc/rc.local

#!/bin/bash
docker start [container_name]
ovs-docker del-port br0 veno1 [containerName]
ovs-docker add-port br0 veno1 [container_name] --ipaddress=---your docker ip---/24 --gateway=---gateway ip---
```

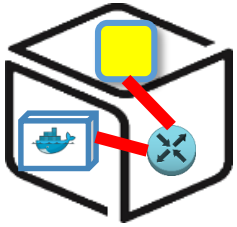
Whenever NUC is rebooted,
network configuration of Docker container is initialized
by executing commands in rc.local

#10 – Check connectivity: VM & Container



#10 - Check connectivity: VM & Container

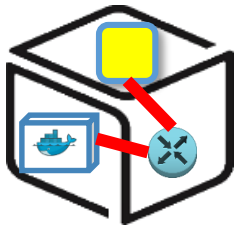
-Check connectivity with ping command



```
root@nuc:/usr/bin# ovs-docker add-port br0 eth0 docker1 --ipaddress=210.125.     /24 --gateway=210.125
root@nuc:/usr/bin# docker attach docker1

root@b8c3bab8204b:/# ifconfig
eth0      Link encap:Ethernet  HWaddr ae:e5:9c:cc:88:b7
          inet addr:210.125        Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::ace5:9cff:fecc:88b7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:120 errors:0 ___pped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8842 (8.8 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@b8c3bab8204b:/# ping google.com
PING google.com (216.58.221.238) 56(84) bytes of data.
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=1 ttl=52 time=41.3 ms
64 bytes from hkg07s21-in-f14.1e100.net (216.58.221.238): icmp_seq=2 ttl=52 time=41.3 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 41.306/41.343/41.380/0.037 ms
```

#10 - Check connectivity: VM & Container

-Check connectivity with ping command



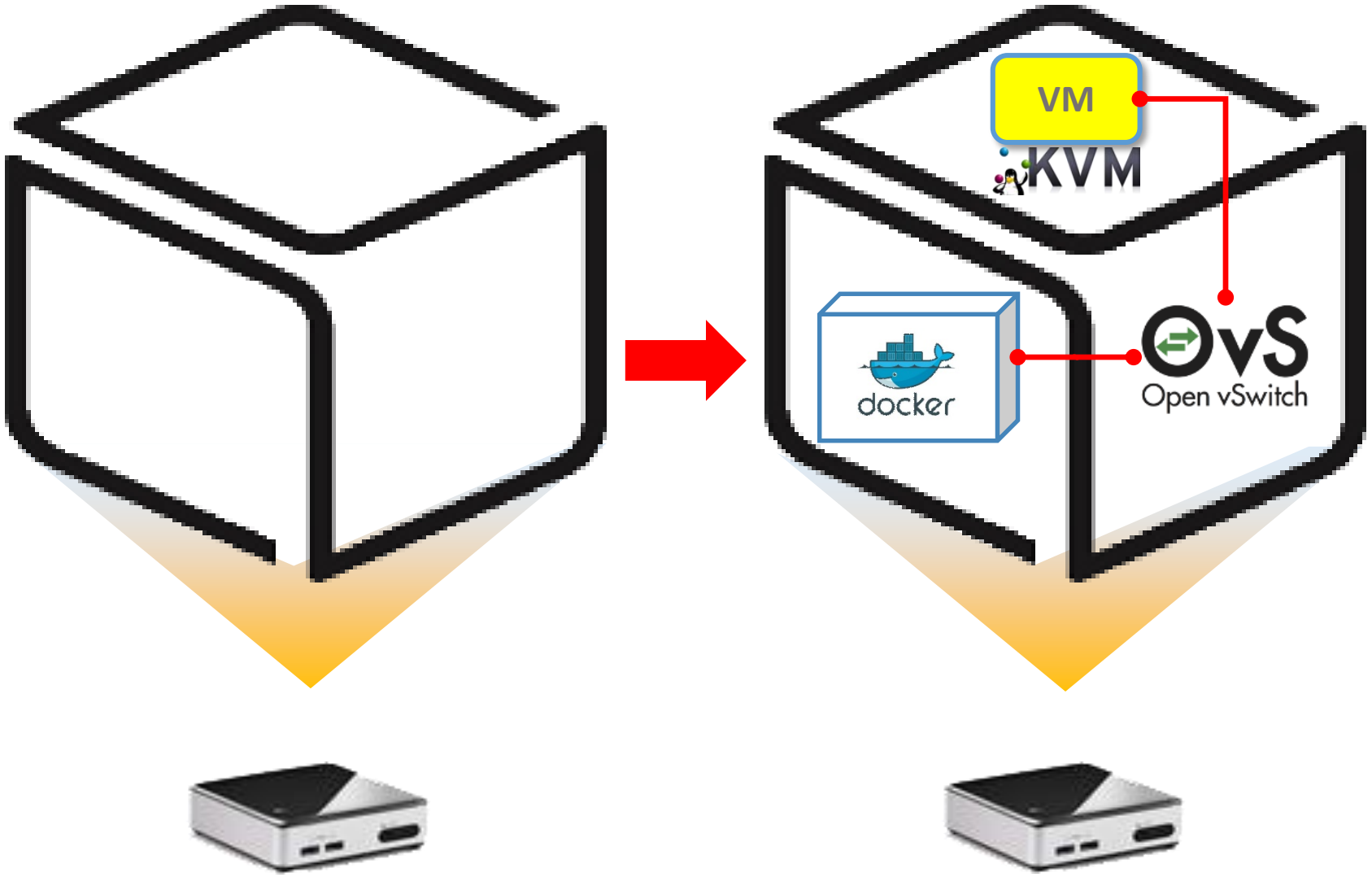
```
root@b8c3bab8204b:/# ifconfig
eth0      Link encap:Ethernet  HWaddr a2:86:d9:c2:33
          inet addr:192.168.     Bcast:0.0.0.0  Mask
          inet6 addr: fe80::a086:d9ff:fec2:337b/64 S
          UP BROADCAST RUNNING MULTICAST  MTU:1500
          RX packets:136 errors:0 dropped:0 overruns
          TX packets:13 errors:0 dropped:0 overruns:
          collisions:0 txqueuelen:1000
          RX bytes:10448 (10.4 KB)  TX bytes:1043 (1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@b8c3bab8204b:/# ping google.com
PING google.com (216.58.221.238) 56(84) bytes of dat
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221
64 bytes from hkg07s21-in-f238.1e100.net (216.58.221
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, t
rtt min/avg/max/mdev = 41.376/41.380/41.384/0.004 ms
root@b8c3bab8204b:/# ping 192.168.
PING 192.168.0.2 (192.168.     56(84) bytes of data.
64 bytes from 192.168.    : icmp_seq=1 ttl=64 time=1.
64 bytes from 192.168.    : icmp_seq=2 ttl=64 time=0.
64 bytes from 192.168.    : icmp_seq=3 ttl=64 time=0.
^C
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, t
rtt min/avg/max/mdev = 0.651/1.028/1.519/0.365 ms
root@b8c3bab8204b:/#
```

```
vbox@vFunction:~$
vbox@vFunction:~$
vbox@vFunction:~$
vbox@vFunction:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr ee:ee:ee:ee:ee:01
          inet addr:192.168.     Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ecee:eeff:feee:ee01/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18857 errors:0 dropped:0 overruns:0 frame:0
          TX packets:69 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1323453 (1.3 MB)  TX bytes:3507 (3.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:38 errors:0 dropped:0 overruns:0 frame:0
          TX packets:38 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3512 (3.5 KB)  TX bytes:3512 (3.5 KB)

vbox@vFunction:~$ ping 192.168
PING 192.168.0.3 (192.168.     56(84) bytes of data.
64 bytes from 192.168.     icmp_seq=1 ttl=64 time=0.872 ms
64 bytes from 192.168.     icmp_seq=2 ttl=64 time=0.590 ms
64 bytes from 192.168.     icmp_seq=3 ttl=64 time=0.585 ms
64 bytes from 192.168.     icmp_seq=4 ttl=64 time=0.573 ms
^C
--- 192.168.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.573/0.655/0.872/0.125 ms
vbox@vFunction:~$
```

Docker container

KVM VM

Box Lab: Final Goal (Recap)



Lab Review

Lab Summary

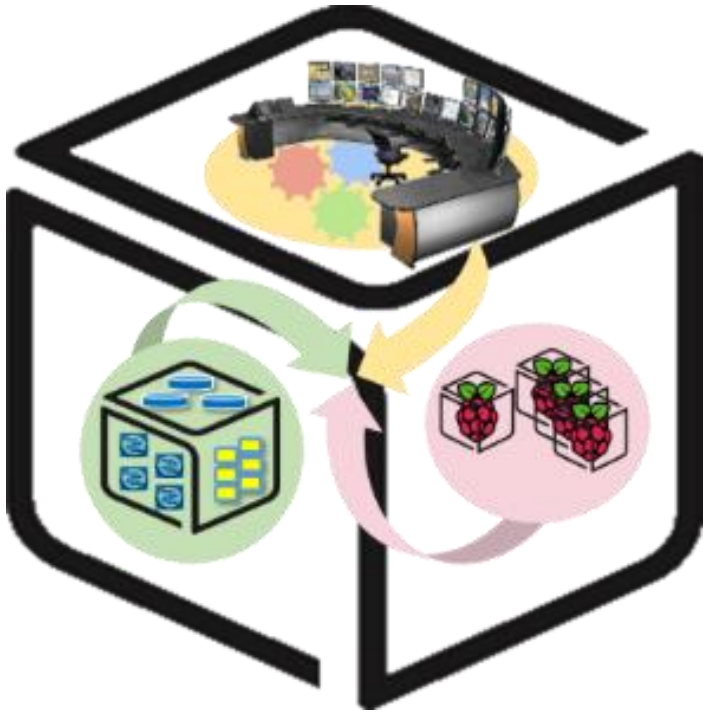
With Box Lab, you have experimented

1. How to install and configure **Linux OS** into Box (i.e., computer).

2. How to install and configure **OVS (Open vSwitch) virtual switch** inside a Linux Box and configure it.

3. How to create **VMs and Docker containers** inside a Linux Box and then **inter-connect** each of them together and to the Internet.

Thank You for Your Attention
Any Questions?



mini@smartx.kr