Juniper vJunos-switch Deployment on KVM

Author : Ridha Hamidi, Sr TME Manager, Juniper Networks, Cloud-Ready Data Center

Summary

Juniper released a new virtual test product named vJunos-switch, based on the traditional Junos OS, and targeted for campus and data center switching use cases.

In this post, we share our experience with deploying and using this new test product on KVM, and we do not aim at covering all cases, nor claiming that following our instructions will guarantee you will successfully deploy and use vJunos-switch on your environments. We expect that some of the instructions below will not apply to your particular case, but hopefully it will cover the majority of the cases.

Abstract

KVM is one of the most popular free virtualized environments in the community, alongside EVE-NG and GNS3. Therefore, it goes without saying that any virtual appliance must be supported on KVM, and must have clear instructions on how to deploy it in this environment.

Introduction

In this post, we provide step-by-step instructions to help users:

- Prepare their KVM environment for vJunos-switch deployment
- Deploy vJunos-switch
- Troubleshoot some of the most common deployment issues
- Build a simple EVPN-VXLAN topology using multiple vJunos-switch instances
- Verify your work

We will try to provide comprehensive explanations about the procedure and explain all the steps. However, for the sake of brevity, we will not address a few topics that might be of interest for some users, like using vJunos-switch with ZTP. This can be the topic of a separate post in the future.

Let's get started.

Prepare your Environment

The server we used in our deployment has the following specs:

- Server: Supermicro SYS-220BT-HNC9R
- CPUs: 128 x Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz
- RAM: 256 GB DDR4
- SSD: 500 GB
- OS: Ubuntu 20.04.5 LTS

We will first verify that this server supports virtualization, then we will install KVM components.

Update packages

```
user@host:~$ sudo apt-get update && sudo apt-get upgrade -y
<output truncated>
```

Check if the Server Supports Hardware Virtualization

```
user@host:~$ grep -Eoc '(vmx|svm)' /proc/cpuinfo
128

user@host:~$
```

In our case, our server has 128 CPUs that have “vmx” or “svm” flags enabled to support virtualization

Check if VT is Enabled in the BIOS

This is done by installing and using the “kvm-ok” tool, which is included in the cpu-checker package.

```
user@host:~$ sudo apt-get install cpu-checker -y
<output truncated>

user@host:~$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
user@host:~$
```

Now that we have checked that the server is ready, we can proceed and install KVM.

Install KVM

There are several packages that need to be installed, some are mandatory, and some are optional

- qemu-kvm: software that provides hardware emulation for the KVM hypervisor.
- libvirt-bin: software for managing virtualization platforms.
- bridge-utils: a set of command-line tools for configuring ethernet bridges.
- virtinst : a set of command-line tools for creating virtual machines.
- virt-manager: provides an easy-to-use GUI interface and supporting command-line utilities for managing virtual machines through libvirt.

We included all mandatory and optional packages in the same install command

```
user@host:~$ sudo apt-get install qemu-kvm bridge-utils virtinst virt-manager -y

<output truncated>
```

Verify installation

```
user@host:~$ sudo systemctl is-active libvirtd
active
user@host:~$


user@host:~$ virsh version
Compiled against library: libvirt 6.0.0
Using library: libvirt 6.0.0
Using API: QEMU 6.0.0
Running hypervisor: QEMU 4.2.1
user@host:~$
```

```
user@host:~$ apt show qemu-system-x86

Package: qemu-system-x86
Version: 1:4.2-3ubuntu6.24
Priority: optional
Section: misc
Source: qemu
Origin: Ubuntu
<output truncated>
```

Now that KVM is installed and up and running, we can proceed and deploy vJunos-switch.

Deploy vJunos-switch on KVM

Pre-requisite Tips

There are a few pieces of information to know about Juniper vJunos-switch before you start building your virtual lab. These are the most important ones:

- vJunos-switch is not an official Juniper Networks product, it is a test product, so it is neither sold nor officially supported by Juniper's TAC, hence the reason for writing this post. If you run into any issues, you can only rely on yourself and the Juniper community. Juniper Networks makes vJunos-switch available for free to download and use with no official support.
- Use vJunos-switch for feature testing only.
- Do not use vJunos-switch for any scaling or performance tests.
- Do not use vJunos-switch in production environments.
- It is recommended to join this Juniper Networks community if you need further support Juniper Labs Community.
- vJunos-switch instance is a nested virtual appliance, that is composed of one single VM that nests both Control and Data Planes, hence vJunos-switch is limited to deployment on bare metal servers only, so do not attempt to deploy vJunos-switch inside another VM, like on EVE-NG running as a VM in another virtualized environment, like ESXi. Doing so will lead to very slow behavior and very poor performance, if at all it works, so do so at your own risk.
- Default login of vJunos-switch is root with no password.
- vJunos-switch must be provisioned with at least one vNIC for the management interface, and as many vNICs as there are data plane interfaces, up to 96 max, and 64 only in the current release 23.1R1.8.
- it is our experience that vNICs are more reliable if you configure them to use virtio driver. We have not experienced any issue when using this driver, instead of the other ones, like e1000.
- There is no license to use vJunos-switch, so all features should work without entering any license key, even though some features will trigger warnings about missing licenses. You can ignore those warning and move on.
- You do not need to have an account with Juniper support to have access to the download page. You only need to agree to the license terms to start downloading the image.

Now that all these notes are read and well understood, we're ready to deploy our first vJunos-switch, knowing all limitations and restrictions.

Download vJunos-switch Image

vJunos-switch image can be downloaded from the official Juniper support page https://support.juniper.net/support/downloads/?p=vjunos.
vJunos-switch documentation can be found at this link https://www.juniper.net/documentation/product/us/en/vjunos-switch.

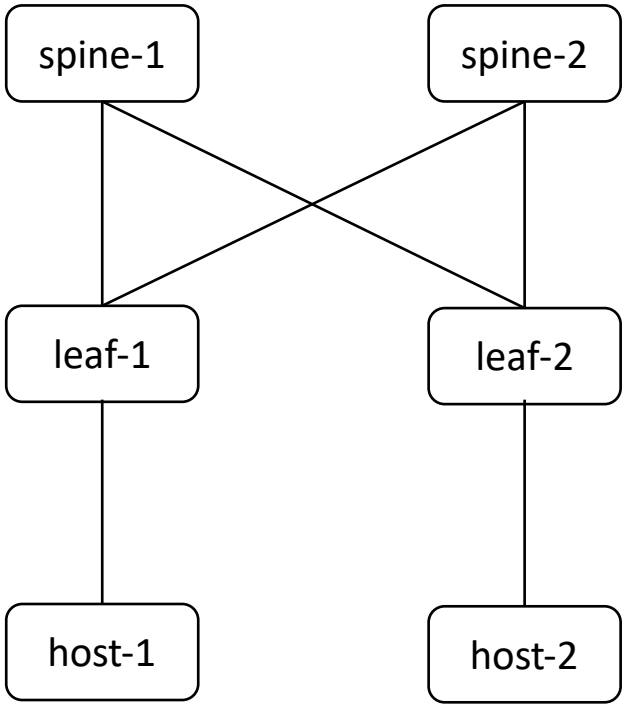
At the time of writing this post, the latest vJunos-switch release is 23.1R1.8.

Note that in addition to vJunos-switch images, to build the topology that we used in this post, you will also need a Linux image to deploy host VMs to be connected to vJunos-switch instances for connectivity testing purposes. The exact type and version of these Linux instances are not important, but if you want to have a good user experience, look for a light weight Linux distribution that supports installing packages for protocols like LLDP and LACP. For this post, we did not need such protocols, so we used cirros-0.5.2 from https://download.cirros-cloud.net/.

Deploy vJunos-switch Instances

For this post, we will deploy the following fabric topology by using shell scripts.

We will deploy the following topology



Given that the downloaded vJunos-switch files are disk images (.qcow2) you must make as many copies as there are vJunos-switch instances to avoid attaching all vJunos-switch instances to the same disk image. This will take up some disk space, so plan your setup accordingly. We will put these disk image copies under the default directory /var/lib/libvirt/images. The following bash script will help make these copies.

(copy-images.sh)

```
#!/bin/bash
sudo cp cirros-0.5.2-x86_64-disk.img /var/lib/libvirt/images/host-1.img
```

```
sudo cp cirros-0.5.2-x86_64-disk.img /var/lib/libvirt/images/host-2.img
sudo cp vjunos-switch-23.1R1.8.qcow2 /var/lib/libvirt/images/vjunos-switch-
23.1R1.8.qcow2
sudo qemu-img create -F qcow2 -b /var/lib/libvirt/images/vjunos-switch-
23.1R1.8.qcow2 -f qcow2 /var/lib/libvirt/images/leaf-1.qcow2
sudo qemu-img create -F qcow2 -b /var/lib/libvirt/images/vjunos-switch-
23.1R1.8.qcow2 -f qcow2 /var/lib/libvirt/images/leaf-2.qcow2
sudo qemu-img create -F qcow2 -b /var/lib/libvirt/images/vjunos-switch-
23.1R1.8.qcow2 -f qcow2 /var/lib/libvirt/images/spine-1.qcow2
sudo qemu-img create -F qcow2 -b /var/lib/libvirt/images/vjunos-switch-
23.1R1.8.qcow2 -f qcow2 /var/lib/libvirt/images/spine-2.qcow2
```

Using “qemu-img create” instead of a simple “cp” allows to have much smaller disk images, as shown below, which is good if you have a limited disk space.

```
user@host:~$ sudo ls -la /var/lib/libvirt/images/
sudo: unable to resolve host host
total 5716108
drwx--x--x 2 root         root       4096 Mar  3 00:17 .
drwxr-xr-x 7 root         root       4096 Feb 26 03:40 ..
-rw-r--r-- 1 libvirt-qemu kvm    36241408 Mar  3 00:23 host-1.img
-rw-r--r-- 1 libvirt-qemu kvm    36241408 Mar  3 00:36 host-2.img
-rw-r--r-- 1 libvirt-qemu kvm   502726656 Mar  3 19:24 leaf-1.qcow2
-rw-r--r-- 1 libvirt-qemu kvm   495648768 Mar  3 19:24 leaf-2.qcow2
-rw-r--r-- 1 libvirt-qemu kvm   490799104 Mar  3 19:24 spine-1.qcow2
-rw-r--r-- 1 libvirt-qemu kvm   509935616 Mar  3 19:24 spine-2.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  3781951488 Mar  3 00:17 vjunos-switch-23.1R1.8.qcow2
user@host:~$
```

We will start by creating the networks of the above topology first then the create the virtual machines after, because this way, VMs can be effectively connected to the appropriate networks immediately when they boot up. It is our experience that when we create the VMs first, then the networks after, we need to reboot the VMs once more for connections to networks to take effect.

To deploy a network, we start by creating its xml definition file. The following is a sample network xml file

(leaf1-spine1.xml)

```
<network>
  <name>leaf1-spine1</name>
  <bridge stp='off' delay='0'/>
</network>
```

then we execute the following commands to create a persistent network, and configure it to start automatically when libvirt daemon is restarted

```
user@host:~$ virsh net-define leaf1-spine1.xml
Network leaf1-spine1 defined from leaf1-spine1.xml

user@host:~$ virsh net-start leaf1-spine1
Network leaf1-spine1 started

user@host:~$ virsh net-autostart leaf1-spine1
Network leaf1-spine1 marked as autostarted

user@host:~$ virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 leaf1-spine1         active     yes           yes
```

```
user@host:~$
```

Note that we used "net-define" and not "net-create" because the former makes the network persistent, whereas the latter makes it transient, that is non persistent across reboots.
The reverse commands to delete the above network are the following:

```
user@host:~$ virsh net-destroy leaf1-spine1
Network leaf1-spine1 destroyed

user@host:~$ virsh net-undefine leaf1-spine1
Network leaf1-spine1 has been undefined

user@host:~$
```

The previous commands will preserve the xml definition file, so you can edit it again and start over.

We used the following bash script to create all the networks we need in this lab setup

(create-networks.sh)

```
#!/bin/bash
echo "<network>" > macvtap.xml
echo "  <name>macvtap</name>" >> macvtap.xml
echo "  <forward mode='bridge'>" >> macvtap.xml
echo "    <interface dev='eno1'/>" >> macvtap.xml
echo "  </forward>" >> macvtap.xml
echo "</network>" >> macvtap.xml
virsh net-define macvtap.xml
virsh net-start macvtap
virsh net-autostart macvtap
for network in leaf1-host1 leaf2-host2 leaf1-spine1 leaf1-spine2 leaf2-spine1
leaf2-spine2
do
    echo "<network>" > $network.xml
    echo "  <name>$network</name>" >> $network.xml
    echo "  <bridge stp='off' delay='0'/>" >> $network.xml
    echo "</network>" >> $network.xml
    virsh net-define $network.xml
    virsh net-start $network
    virsh net-autostart $network
done
```

After completing the previous task for all networks with your preferred method, you should see the following output

```
user@host:~$ virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 leaf1-host1          active     yes           yes
 leaf1-spine1         active     yes           yes
 leaf1-spine2         active     yes           yes
 leaf2-host2          active     yes           yes
 leaf2-spine1         active     yes           yes
 leaf2-spine2         active     yes           yes
 macvtap              active     yes           yes

user@host:~$
```

Deploy vJunos-switch

There are multiple ways to deploy vJunos-switch instances on KVM, one can name at least these three:

- virt-manager : deploy all VMs by using KVM GUI
- virsh define/create : these methods require creating an XML definition file of each VM
- virt-install : CLI based method. One needs only to specify the deployment parameters of VMs

virt-manager procedure works fine for small setups. The GUI is very intuitive, which makes this method less error prone. However, like most GUI-based tools, it does not scale well if you need to create a large number of VMs.

virsh define/create require creating an xml definition file for each VM. It is highly recommended that you start with an existing XML file from a previously created and working VM, and not start from scratch, if you use virsh define/create. If you do not have any XML file to start with, you can create a VM by using the virt-manager, then copy its XML file located under /etc/libvirt/qemu/, or generate it by using the command

```
virsh-dumpxml vm_name > vm_name.xml
```

Note that Juniper provides a vJunos-switch xml file on the Downloads page, alongside a vJunos-switch meta disk script for its initial configuration.

Even though all 3 methods would work just fine, depending on the user's familiarity with each method, in this post, we're going to use virt-install, because we can put all commands inside a shell script, and repeat the operation if anything is not working as expected. Like with any new product, it might take a few tries before you get everything right with all the parameters.

Below is a sample virt-install command to deploy one of the vJunos-switch VM:

```
virt-install \
     --name leaf-1 \
     --vcpus 4 \
     --ram 5120 \
     --disk path=/var/lib/libvirt/images/leaf-1.qcow2,size=10 \
     --os-variant generic \
     --import \
     --autostart \
     --noautoconsole \
     --nographics \
     --serial pty \
     --cpu IvyBridge,+vmx \
     --sysinfo smbios,system_product=VM-VEX \
     --network network=macvtap,model=virtio \
     --network network=leaf1-spine1,model=virtio \
     --network network=leaf1-spine2,model=virtio \
     --network network=leaf1-host1,model=virtio
```

Below is a sample virt-install command to deploy one of the Cirros host VMs:

```
virt-install \
     --name host-1 \
     --vcpus 1 \
     --ram 1024 \
     --disk path=/var/lib/libvirt/images/host-1.img,size=10 \
     --os-variant generic \
     --import \
     --autostart \
     --noautoconsole \
```

```
    --nographics \
    --serial pty \
    --network network=macvtap,model=virtio \
    --network network=leaf1-host1,model=virtio
```

A few comments about the command above:

- "`--serial pty`" allows you to access the guest VM's console from the host by using "`virsh console <vm_name>`". Another alternative to allow access to guest VMs from the host is via telnet by specifying, for example "`--serial tcp,host=:4001,mode=bind,protocol=telnet`", where 4001 is a tcp port that is specific to the VM, so it must be different for each guest VM.
- The first interface of the VM is connected to "macvtap" bridge that connects to host's interface eno1. This way the VM will be connected to the same management network as the host itself, so we can reach it directly without jumping to the host. Other alternatives to how to manage the guest VM include the following:
    - "`--network type=direct,source=eno1,source_mode=bridge,model=virtio`": this works the same way than "macvtap" bridge
    - "`--network network=`**`default,`**`model=virtio`" : this way, the VM will be connected to KVM's internal default network and will get an IP address via DHCP from the default 192.168.122.0/24 subnet. To make the VM reachable from outside, we need to configure the host for port forwarding.

Note that your host's management interface name can be different than eno1 above, like eth0 or something else, so please update the script accordingly.

We used a bash script to create the VMs needed in this lab setup. The script is not provided here for brevity, but it executes the above virt-install command for as many times as there are VMs with the specific parameters and interfaces.

Note that we provisioned each vJunos-switch with 4 vCPU and 5 GB of RAM.

You can delete a VM by using the following commands, for example

```
virsh shutdown leaf-1
virsh undefine leaf-1
sudo rm /var/lib/libvirt/images/leaf-1.qcow2
```

Once all VMs are deployed, you should see the following output:

```
user@host:~$ virsh list
Id    Name                           State
----------------------------------------------------
42    leaf-1                         running
43    leaf-2                         running
44    spine-1                        running
45    spine-2                        running
46    host-1                         running
47    host-2                         running

user@host:~$
```

At this point, all VMs should be up and running, and should be reachable directly from the outside without jumping on the host. However, vJunos-switch management interfaces run DHCP by default, so we do not know at this point what IP addresses have been assigned to

nJunos instances. To that end, we must console to the instances either from the host by using “virsh console” or telnet to the specific port, as explained above, or by using virt-manager.
The vJunos-switch instances should reach each other once we complete the basic Junos configurations. Let's verify that.

Verifications

Try accessing the console of one of the vJunos-switch instances by using the following command

```
user@host:~$ virsh console leaf-1
Connected to domain leaf-1
Escape character is ^]



FreeBSD/amd64 (Amnesiac) (ttyu0)

login: root
Last login: Fri Mar  3 00:29:56 on ttyu0

--- JUNOS 23.1R1.8 Kernel 64-bit  JNPR-12.1-20230307.3e7c4b6_buil
root@:~ #
```

The default credentials are "root" and no password.

Enable Junos CLI and verify that the dataplane is online

```
root@:~ # cli
root> show chassis fpc

                     Temp  CPU Utilization (%)   CPU Utilization (%)  Memory
Utilization (%)
Slot State            (C)  Total  Interrupt      1min   5min   15min  DRAM (MB)
Heap     Buffer
  0  Online           Testing   4          0        3      3      3    1023       19
0
  1  Empty
  2  Empty
  3  Empty
  4  Empty
  5  Empty
  6  Empty
  7  Empty
  8  Empty
  9  Empty
 10  Empty
 11  Empty

root>
```

Verify that 10 "ge" interfaces are present.

```
{root> show interfaces ge* terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.16386          up    up
ge-0/0/1                up    up
ge-0/0/1.16386          up    up
ge-0/0/2                up    up
```

```
ge-0/0/2.16386          up    up
ge-0/0/3                up    down
ge-0/0/3.16386          up    down
ge-0/0/4                up    down
ge-0/0/4.16386          up    down
ge-0/0/5                up    down
ge-0/0/5.16386          up    down
ge-0/0/6                up    down
ge-0/0/6.16386          up    down
ge-0/0/7                up    down
ge-0/0/7.16386          up    down
ge-0/0/8                up    down
ge-0/0/8.16386          up    down
ge-0/0/9                up    down
ge-0/0/9.16386          up    down

root>
```

A vJunos-switch instance comes up with 10 ge-x/x/x interfaces by default, but you can configure it with up to 96 ge-x/x/x interfaces by using the following command:

```
set chassis fpc 0 pic 0 number-of-ports 96
```

At the time of writing this post, vJunos-switch version 23.1R1.8 supports only 64 interfaces. If you try to configure more interfaces, you will get the following error message:

```
[edit]
root@leaf-1# ...0/0/64 unit 0 family ethernet-switching
error: port value outside range 0..63 for '64' in 'ge-0/0/64': ge-0/0/64

[edit]
root@leaf-1#
```

This limitation will be removed in the subsequent releases.

Try accessing the console of one of the host VMs

```
user@host:~$ virsh console host-1
Connected to domain host-1
Escape character is ^]

login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
cirros login: cirros
Password:
$
```

If you used an Ubuntu image for the host VMs, you may not have access to the console with "`virsh console`". If that's the case, access the console with virt-manager and make the following changes in the guest VM:

```
sudo systemctl enable serial-getty@ttyS0.service
sudo systemctl start serial-getty@ttyS0.services
```

edit file /etc/default/grub of the guest VM and configure the following lines

```
GRUB_CMDLINE_LINUX_DEFAULT="console=tty0 console=ttyS0"
GRUB_TERMINAL="serial console"
```

then

```
sudo update-grub
```

At this point, you should be able to access the Ubuntu VMs with "`virsh console`".

Now that all looks good and we're ready to start configuring our devices to build the EVPN-VXLAN topology shown above.

Configuration

Base vJunos-switch Configuration

When a vJunos-switch instance comes up, it has a default configuration that needs to be cleaned up before we proceed further. For example, the default configuration is ready for ZTP, which we will not address in this post.

The following configuration should be the bare minimum needed to start using a vJunos-switch instance, and you can delete everything else :

```
[edit]
root@leaf-1# show
## Last changed: 2023-04-14 01:31:59 UTC
version 23.1R1.8;
system {
    host-name leaf-1;
    root-authentication {
        encrypted-password "*****"; ## SECRET-DATA
    }
    services {
        ssh {
            root-login allow;
        }
        netconf {
            ssh;
        }
    }
    syslog {
        file interactive-commands {
            interactive-commands any;
        }
        file messages {
            any notice;
            authorization info;
        }
    }
}
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                dhcp {
                    vendor-id Juniper-ex9214-VM64013DB545;
                }
            }
        }
    }
}
protocols {
    lldp {
        interface all;
        interface fxp0 {
            disable;
```

```
        }
    }
}

[edit]
root@leaf-1#
```

Once all vJunos-switch instances have the minimum configuration above entered and committed, let us verify that the topology is built properly by verifying LLDP neighborship.

```
[edit]
root@leaf-1# run show lldp neighbors

[edit]
root@leaf-1#

[edit]
root@leaf-1# run show lldp statistics
Interface    Parent Interface  Received  Unknown TLVs  With Errors  Discarded TLVs
Transmitted  Untransmitted
ge-0/0/0     -                 0         0             0            0
127          0
ge-0/0/1     -                 0         0             0            0
127          0
ge-0/0/2     -                 0         0             0            0
127          0

[edit]
root@leaf-1#
```

The output above shows that LLDP neighborship are not forming, and that the instance is transmitting LLDP packets but is not receiving any. This is expected because, by default, IEEE 802.1D compliant bridges Linux bridges do not forward frames of link local protocols, like LLDP and LACP. For reference, the destination MAC address used by LLDP is 01-80-C2-00-00-**0E**.

Let's checking the default value of the Group Forwarding Mask of one of the bridges

```
user@host:~$ cat /sys/class/net/virbr1/bridge/group_fwd_mask
0x0
user@host:~$
```

The zero value means that this bridge does not forward any link local protocol frames. To change the bridge's behavior and force it to forward LLDP frames we need to set the 15th bits of this 16-bit mask, specified by the rightmost 0xE in the MAC address, that is writing the hex value 0x4000, or decimal value 2^14=16,384, to `group_fwd_mask`. Let's try it on all bridges of this host.

We used the following shell script to accomplish that. Please adjust bridge and interface numbers to your specific case ; use "`brctl show`" command to get the number of virbrX bridges and vnetX interfaces

```
user@host:~$ brctl show
bridge name  bridge id           STP enabled  interfaces
PFE_LINK             8000.000000000000   no
RPIO_LINK            8000.000000000000   no
docker0              8000.024236eb3574   no
virbr0       8000.525400499cd3   yes          virbr0-nic
virbr1       8000.525400d33e1f   no           virbr1-nic
                                             vnet17
```

```
                                                vnet2
virbr10             8000.525400f3b32f   no              virbr10-nic
                                                vnet3
                                                vnet6
virbr11             8000.525400515275   no              virbr11-nic
                                                vnet4
                                                vnet5
virbr12             8000.525400fc6995   no              virbr12-nic
                                                vnet11
                                                vnet14
virbr13             8000.5254001353b6   no              virbr13-nic
                                                vnet12
                                                vnet13
virbr2        8000.52540000b6ef   no              virbr2-nic
                                                vnet18
virbr3        8000.525400dc812d   no              virbr3-nic
                                                vnet0
                                                vnet9
virbr4        8000.525400703dbd   no              virbr4-nic
                                                vnet1
                                                vnet15
virbr5        8000.525400d5048a   no              virbr5-nic
                                                vnet10
                                                vnet7
virbr7        8000.5254004c3aa1   no              virbr7-nic
                                                vnet16
                                                vnet8
virbr8        8000.525400fb9b9a   no              virbr8-nic
virbr9        8000.525400c7766f   no              virbr9-nic
user@host:~$
```

overwrite-mask.sh

```
#!/bin/bash
for i in {0..13}
do
    echo 0x4000 > /sys/class/net/virbr$i/bridge/group_fwd_mask
done
```

At this point, LLDP should be working fine on all vJunos-switch instances, as shown below

```
[edit]
root@leaf-1# run show lldp statistics
Interface    Parent Interface  Received  Unknown TLVs  With Errors  Discarded TLVs
Transmitted  Untransmitted
ge-0/0/0     -                 10        0             0            0
170          0
ge-0/0/1     -                 10        0             0            0
169          0
ge-0/0/2     -                 0         0             0            0
167          0

[edit]
root@leaf-1# run show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info
System Name
ge-0/0/0           -                   2c:6b:f5:19:f5:c0   ge-0/0/0
spine-1
ge-0/0/1           -                   2c:6b:f5:a6:fc:c0   ge-0/0/0
spine-2

[edit]
root@leaf-1#
```

```
[edit]
root@leaf-1# run show lldp neighbors
```

```
Local Interface    Parent Interface    Chassis Id          Port info
System Name
ge-0/0/0           -                   2c:6b:f5:1a:65:c0   ge-0/0/0
spine-1
ge-0/0/1           -                   48:cd:91:2e:05:d5   et-0/0/0
spine-2

[edit]
root@leaf-1#
```

Note that LLDP neighborship is not forming between leaf-1 and host-1, and that's because Linux does not include LLDP package by default. It can be installed on some Linux distributions, but not on Cirros, that we used here.

```
root@leaf-1> show system information
Model: ex9214
Family: junos
Junos: 23.1R1.8
Hostname: leaf-1

root@leaf-1>
```

At this point, all vJunos-switch instances should be deployed and connected properly and you should be able to start playing with your topology.

Have fun !

Conclusions

In this post, we shared the step to successfully deploy vJunos-switch virtual appliance. Our purpose was to provide all the details to avoid running into issues that might cause long troubleshooting sessions. We hope we achieved this goal.

Useful links

Juniper Networks software downloads -
https://support.juniper.net/support/downloads/?p=vjunos
Juniper vJunos-switch product page
https://www.juniper.net/documentation/product/us/en/vjunos-switch

Acknowledgements

Special thanks to the following individuals who helped in building and troubleshooting the vJunos-switch setup we used for this post, and who also helped reviewing this post:

- Aninda Chatterjee
- Art Stine
- Kaveh Moezzi
- Shalini Mukherjee
- Yogesh Kumar