Juniper Networks | Engineering Simplicity

Virtual Switches Managed by Mist Cloud—Network Configuration Example (NCE)

Published
2025-07-15

Table of Contents

Virtual Switches Managed by Mist Cloud —Network Configuration Example (NCE)

Juniper Networks Network Configuration Example (NCE) describes how to configure and deploy Juniper products in a typical use case scenario. In this NCE, you'll find use case scenario with the topology, configuration information, and validation output for the configuration. Read further to plan and optimize your network deployment.

About this Document

This document is to help you to build your own labs using free-of-charge vJunos-switch VM. It contains information about:

- Using this VM in hypervisor environments such as EVE-NG, Ubuntu native KVM (libvirtd) and Proxmox VE.
- How to manage vJunos-switch using the Mist Cloud.
- How to create a customized Junos OS configuration and apply it to the VM when launching.
- How to embed adopt configuration and make the VM automatically appear in the Mist Cloud inventory of an organization.
- Creating Linux network bridges to connect VMs internally and change them after VM launch for higher MTU and passes LLDP, LACP and 802.1X messages.
- Using libvirtd GUI and CLI to launch vJunos-Switch VMs.
- Optimizing Ubuntu KVM servers for faster / more vJunos-switch VMs.
- Much more best practices.

Introduction to the vJunos-switch VM

vJunos-switch is a virtual Junos® OS based platform designed to easily create instant virtual lab topologies for training, demo, proof of concept, script development, configuration generation, and testing the control plane in virtual lab environments.

This is all done without having to build labs with Juniper Networks' physical routers and switches based on Junos OS. vJunos Lab platforms are available at no cost for non-production test environments only and have no time limit—feel free to use them as long as you like.

> **NOTE**: vJunos Labs are for non-production test environments only. These products are exclusively for testing configurations and running low traffic to validate switch topologies. Juniper provides no commercial customer support for this software. Also, Juniper Networks Technical Assistance Center (JTAC) support is not available.

This document supplements the public vJunos-switch documentation with information from users that have already built labs in various hypervisor environments. The focus of this document is to show how to run vJunos-switch VM in each hypervisor environment so that you can reproduce the steps easily. An example can help you understand how to build the lab before you try it yourself. With this document, you will learn how to build the lab using an example.

The key focus of this document is the integration to Mist Cloud as a management tool for vJunos-switch. This allows you to simulate branch topologies as well as campus fabric designs.

In this document, the described hypervisor environments are based on the officially supported Debian or Ubuntu Linux distributions for vJunos-switch:

- EVE-NG is a complete and GUI based solution to build virtual labs.
- Ubuntu KVM with libvirtd-based VM management is a minimum hypervisor environment that can run your vJunos-switch VMs and in parallel execute other services at the same time on the server.
- Proxmox VE is a scalable hypervisor environment designed for data centers.

If you have not selected a particular environment yet, then the table below might help you select the right environment.

Environment	Knowledge Level Required	Linux Bridge Updated	Cluster Option	RAM Usage Optimized	Server Minimum RAM	Integrate Physical Hardware	Network Sniffer
EVE-NG on BMS	Beginner	Yes	Yes (professional version only)	Yes	32 GB	Limited to networks	Wireshark
Ubuntu KVM w. libvirtd	Expert	Needs Script	No	Optional	64 GB or 32 GB with UKSM Kernel	Yes	Local tcpdump

(Continued)

Environment	Knowledge Level Required	Linux Bridge Updated	Cluster Option	RAM Usage Optimized	Server Minimum RAM	Integrate Physical Hardware	Network Sniffer
Proxmox VE	Expert	Needs Script	Yes	No	64 GB	Yes	Local tcpdump

For all these environments, you must have a dedicated bare metal server (BMS). If you want to run multiple vJunos-switch VMs in parallel, then consider having a minimum of 32 GB RAM and 16 vCPUs on your server. CPU oversubscription is possible but must be carefully evaluated as the experience changes throughout the different hardware. We recommend not to oversubscribe more than twice the level of 4 vCPUs per vJunos-switch instance required.

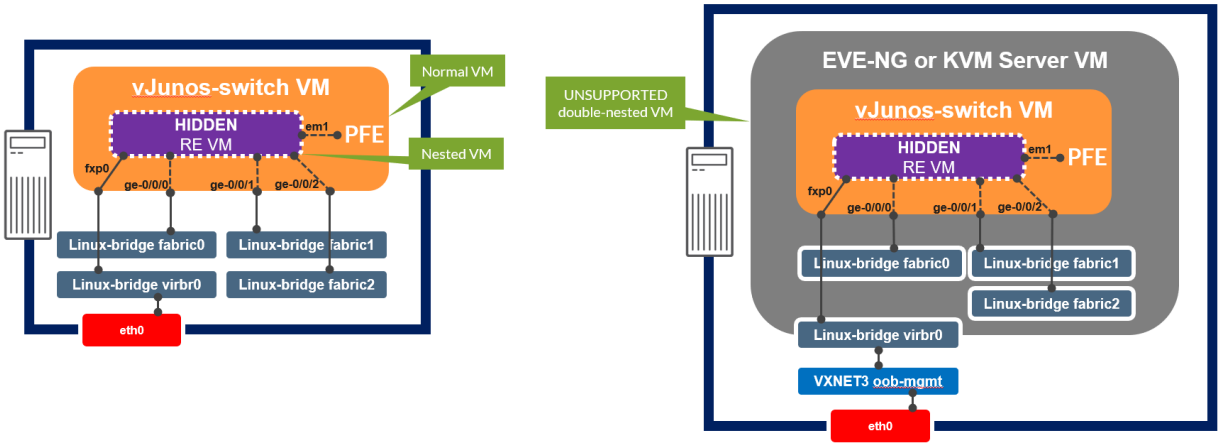
Deployment and Feature Restrictions of vJunos-switch VM

Before you begin, you must:

- Know the vJunos-switch deployment restrictions.
- Review the official limitations per vJunos-switch Release Notes.

The Release Notes states that “Due to its nested architecture, the vJunos-switch cannot be used in any deployments that launch the instances from within a VM.” This means that only a BMS-based deployment is supported. This is because the vJunos-switch VM launches additional VMs internally and connects them to as an easily manageable and deployable single VM externally. But the launched additional VMs are archived in this single VM. This means:

- This document covers the instructions that you need to pass a special CPU flag to the VM to allow this nested operation even on a regular deployment on a supported BMS Platform.
- You cannot deploy or use vJunos-switch in another VM as double-nesting is not supported.



BMS EVE-NG or KVM Server
Only one Level of VM nesting

Hosted (ESXi) Server VM
Double-nesting of VM's
is NOT SUPPORTED!

NOTE: Avoid double-nesting deployment. The start time for these systems might exceed an hour than the usual ~three minutes. They might not find their FCP or create any virtual ge-0/0/x revenue interfaces, and crash frequently.

Also, in the current version, an ungraceful shutdown of a launched vJunos-switch has a high chance to corrupt the internal disk. After a restart, the VM fails to reboot correctly and is unusable. To avoid this situation, we recommend using:

- BMS KVM server—Use `virsh shutdown <VM>` to shut down the VM gracefully. This might take less than 20 seconds. Avoid using `virsh destroy <VM>` to encounter the ungraceful shutdown issue.
- BMS EVE-NG server—EVE-NG does not support a graceful shutdown option. You must connect to the VM console and run the command `request system power-off` before you instruct EVE-NG to stop the VM.

The following are few other known vJunos-switch feature limitations that are asked frequently, and are not explicitly mentioned in the vJunos-switch Release Notes:

- vJunos-switch does not support building Virtual Chassis.
- vJunos-switch does not support VXLAN group-based policy (GBP). You can assign and view GBP-Tags, but the enforcement functions and the traffic blocking does not work.
- When testing DHCP relay in campus fabric, you might see duplicates of the original client requests on the DHCP server. This is because `no-dhcp-flood` option is not enforced on the irb configuration. But the client can still get its lease as the DHCP protocol is robust against these duplicated messages.

You can use physical Juniper switches to evaluate the Junos OS features. You can also build hybrid labs using both physical and virtual switches in a single campus fabric lab, but this is not in the scope of this document.

NOTE: We do not officially support deployment on AMD CPUs. You can try the unofficial tweaks mentioned in chapter "AMD-CPU Unofficial Tweaks" on page 56 but we cannot guarantee success.

Manage vJunos-Switch VMs in Mist Cloud

IN THIS SECTION

When you manage vJunos-switch VMs using Mist Cloud, consider the following needs:

- You must have your own Mist Cloud account to onboard a device.
- Your lab firewall must allow a raw TCP connection to outbound destination port 2200. For more information on enabling your firewall ports to manage the device, see Ports to enable on your firewall - Mist.
- The redirect server cannot identify the device's serial number as the device has not gone through a manufacturing process. Hence, you can't onboard the device using the Claim and ZTP method. Instead, vJunos-switch VMs must use the adopted process to appear in the inventory. For an example of automating this process, see the chapter "Default Junos OS Configuration for vJunos-switch" on page 20.
- vJunos-switch VMs acts as EX9200 line of switches. In any campus fabric designs, you cannot use the physical EX9200 Series switches as access switches. As an exception, the Mist GUI allows virtual EX9200 Series switches so that you can use vJunos-switch VM to simulate as an access switch in a campus fabric lab.
- We recommend using the management interface fxp0 (first VM interface) for vJunos-switch VMs to get a DHCP lease through Mist Cloud. This is required for simulating campus fabric labs but not mandatory for a branch.

- When you launch multiple vJunos-switch VMs, the VMs appear in the Mist Cloud inventory. You must assign each VM to a site individually from this inventory. Assigning multiple VMs at the same time and overloading the server might result in disconnection and the need for VM reboot. This is because of a sudden disk-IO surge when changes are applied to multiple VMs at once.

Difference Between Device ID, MAC Address, and Serial Number

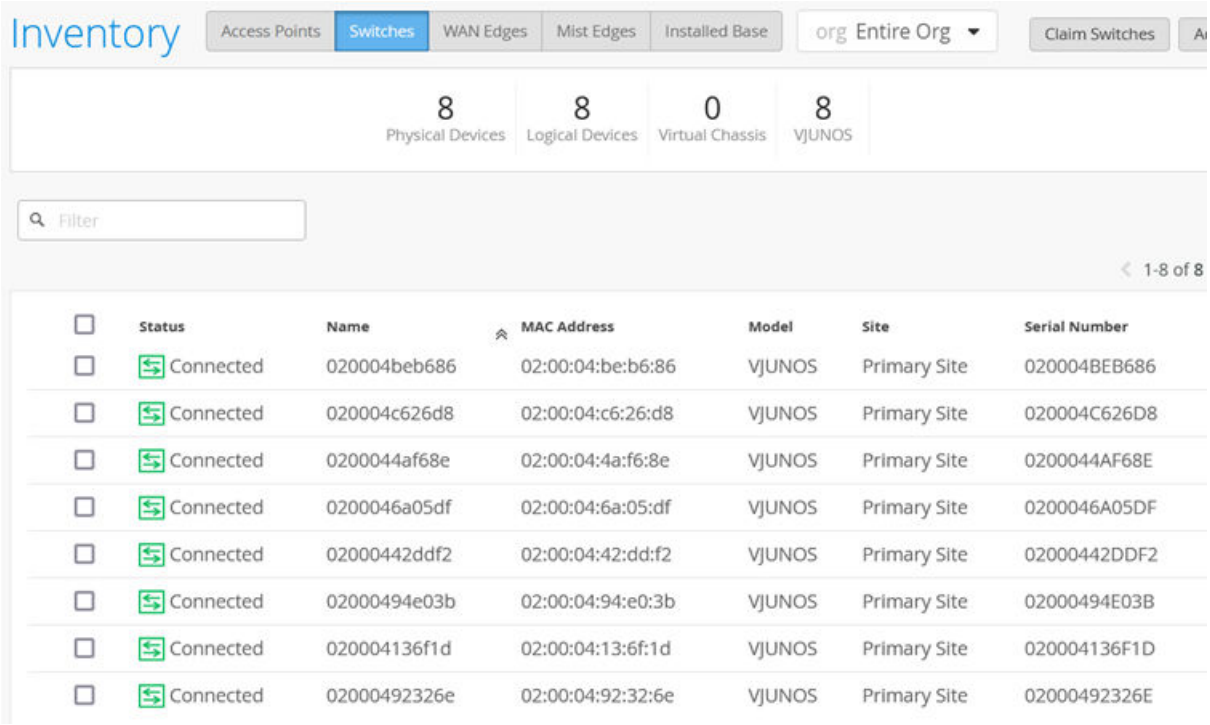
To support and manage virtual devices such as vJunos-switch, the Mist Cloud's internal behavior must be changed. For physical devices, excluding virtual switches, the device ID is same as the device's MAC address. To support the required scale, the device ID was decoupled from the system's MAC address. This is because the randomly generated address did not have required information. Now, the expected behavior is that:

- **Device ID**: Is a unique 6 byte address used for internal system management and reference.
  - You must know the device ID when using API calls.
  - Mist Cloud uniquely assigns device IDs during device adoption to avoid overlaps, even across different organizations.
  - All vJunos-switch device IDs starts with 02:00:04, but this could change.
  - To get the device ID from a vJunos-switch VM, use the following command in the Junos OS configuration: `show configuration | display set | match device-id` as in the example later in this section.
- **MAC-Address**: Is a random MAC address that the vJunos-switch VM generates locally when it starts for the first time.
  - This is the base MAC address, which is incrementally assigned to Ethernet interfaces.
  - This MAC address is reported when you search for LLDP neighbors.
  - This MAC address might not be unique and might overlap with other vJunos-switch VM's in the same lab, but the chances are 1:65536.
  - All vJunos-switch MAC addresses start with `2c:6b:f5`.
  - To get the MAC address from a vJunos-switch VM, use the following command in the Junos OS configuration: `show chassis mac-addresses`. Then, get the reported private base address.
- **Serial Number**: Is a random serial number that the vJunos-switch VM generates locally when it starts for the first time.
  - The serial number has no relevance for vJunos-switch VMs because these are free training VMs and do not need any licenses.

- All vJunos-switch serial numbers start with `VM67`.
- To get the serial number from a vJunos-switch VM, use the following command in the Junos OS configuration: `show chassis hardware`.

After you launch multiple vJunos-switch VMs and run the adoption Junos OS CLI, the VMs appear in the inventory.

In the latest Mist Cloud, the device ID is reported as both MAC address and serial number.



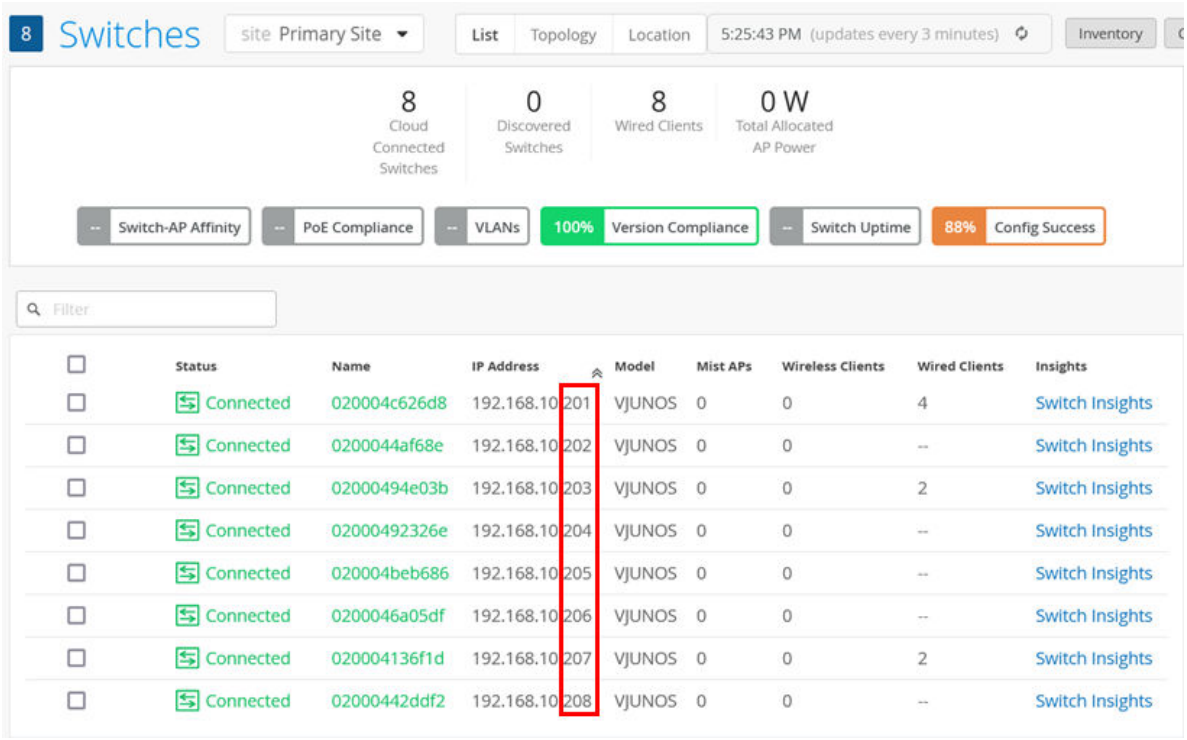
When you assign the VM to a site and allow Mist to manage it, you can use any of the following options:

1. Access Remote Shell through the Mist GUI or use your hypervisor's console to the VM. Once you are on Junos OS CLI, run `show configuration | display set | match device-id` to review the added configuration to the organization ID, which now includes the management MAC address. Review the example below.

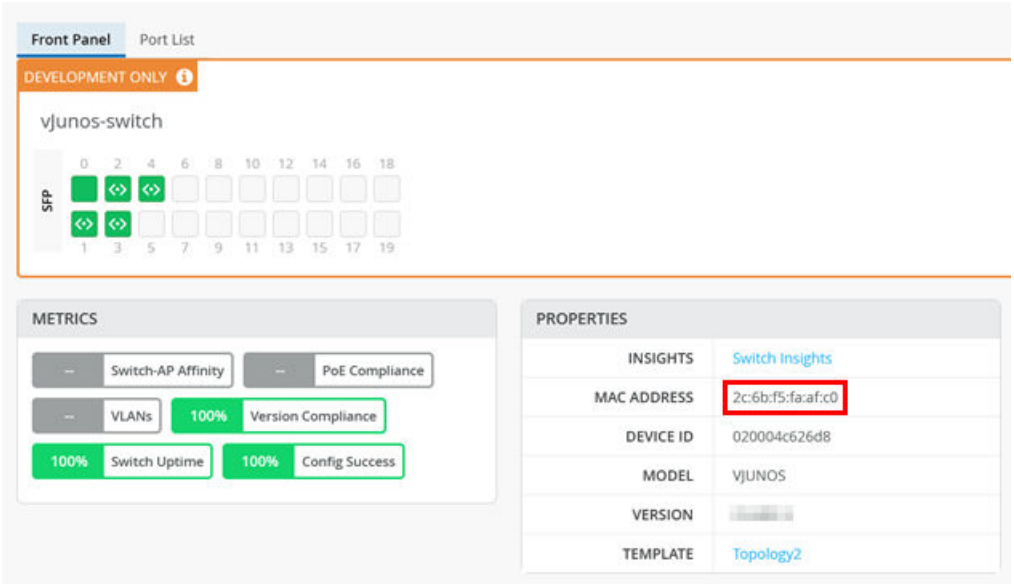
```
mist@switch1> show configuration | display set | match device-id
set system services outbound-ssh client mist device-id abc9254b-bc35-4a09-
a625-681c18f52645.020004c626d8
```

2. Start managing the device. Then, read the IP address of each virtual switch. You can indirectly influence the reported IP address. Each vJunos-switch VM interface indicates in the order, first, out-

of-band management interface fxp0, next ge-0/0/0, then ge-0/0/1, and so on. When you assign a unique MAC address to each first interface, then map the MAC address to an IP address statically in your DHCP server. This allows you to identify the VM by its IP address. For an example DHCP server configuration, see "Preparations Before You Deploy" on page 16.



3. Obtain the private base MAC address to determine the device. As this MAC address is always not unique, you will not find duplicates on multiple virtual devices in your lab. In the switch GUI, you can find the reported MAC address as shown below.



4. Access Remote Shell through the Mist GUI or use your hypervisor's console to the VM to obtain the MAC address information. Once you are on Junos OS CLI, run `show chassis mac-addresses` to review the generated private base address.

```
mist@020004c626d8> show chassis mac-addresses
MAC address information:
  Public base address      2c:6b:f5:fa:a8:00
  Public count             1984
  Private base address     2c:6b:f5:fa:af:c0
  Private count            64

mist@020004c626d8>
```

Knowing each system's private base address helps you to identify the MAC address. In other directly connected systems, this address appears as the **Chassis Id** of reported LLDP neighbors.

```
mist@020004136f1d> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System Name
ge-0/0/2           -                   2c:6b:f5:0d:eb:c0   ge-0/0/3           0200044af68e
ge-0/0/1           -                   2c:6b:f5:fa:af:c0   ge-0/0/3           020004c626d8
```

EVE-NG BMS Environment

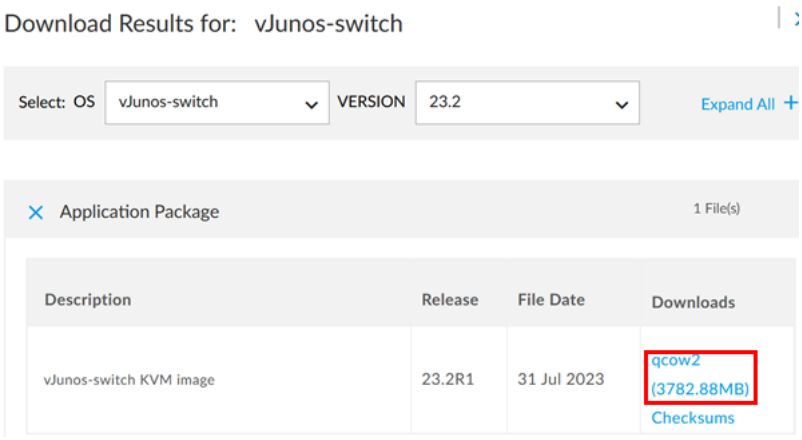
IN THIS SECTION

If you have not installed EVE-NG 5.x or later on a BMS, check the download link to get to the Professional or Community ISOs.

NOTE: Use only the ISO files and install on a BMS. Avoid using the OVA VM images as they would cause the unsupported double-nesting issue.

Copy the VM Image to Your EVE-NG Environment

1. Go to the vJunos-switch download link https://support.juniper.net/support/downloads/?p=vjunos and select the latest image as shown below.



2. Click **copy** to copy the temporary URL, which is valid for 10 minutes.

3. Access SSH to connect to your EVE-NG BMS as root to install the image. Embed the copied URL to `wget "<download-url>"` as shown below.

```
sudo -i
# create a directoy with "vjunosswitch-version" pattern
mkdir /opt/unetlab/addons/qemu/vjunosswitch-23.2
cd /opt/unetlab/addons/qemu/vjunosswitch-23.2
# download the image
wget "https://cdn.juniper.net/software/vJunos-switch/23.2R1/vJunos-switch-23.2R1.14.qcow2?
SM_USER=anon&__gda__=1697405150_8fa46e0fe03d658e6dd6280aef5aa151"
# rename the image to hda.qcow2 NOT virtioa.qcow2
mv "vJunos-switch-23.2R1.14.qcow2?
SM_USER=anon&__gda__=1697405150_8fa46e0fe03d658e6dd6280aef5aa151" hda.qcow2
```

Use `hda.qcow2` as VM image name, which causes integrated development environment (IDE) instead of virtio drivers environment. Else, a custom Junos OS will not work as described in chapter "Default Junos OS Configuration for vJunos-switch" on page 20.

Create Template Files for the VM

Continue your SSH access to your EVE-NG BMS as root. Create two template files for the supported Intel and the unsupported AMD version as shown below. The unique Qemu parameters that are required to start the image are shown in bold below. A Stop-Command in the UI is added `shutdown: 0` for the community version that enables a graceful shutdown. Else, you might run into issues as described in chapter "Deployment and Feature Restrictions of vJunos-switch VM" on page 3.

```
cat <<EOF >/opt/unetlab/html/templates/intel/vjunosswitch.yml
---
type: qemu
description: Juniper vEX Switch
name: vEX
cpulimit: 4
icon: JunipervQFXpfe.png
cpu: 4
ram: 5120
eth_name:
- fxp0
eth_format: ge-0/0/{0-9}
ethernet: 11
console: telnet
shutdown: 0
qemu_arch: x86_64
qemu_version: 5.2.0
qemu_nic: virtio-net-pci
qemu_options: -machine type=pc,accel=kvm -serial mon:stdio -nographic -smbios type=1,product=VM-
VEX -cpu IvyBridge,ibpb=on,md-clear=on,spec-ctrl=on,ssbd=on,vmx=on
...
EOF
cat <<EOF >/opt/unetlab/html/templates/amd/vjunosswitch.yml
---
type: qemu
description: Juniper vEX Switch
name: vEX
cpulimit: 4
icon: JunipervQFXpfe.png
cpu: 4
ram: 5120
eth_name:
- fxp0
```

```
eth_format: ge-0/0/{0-9}
ethernet: 11
console: telnet
shutdown: 0
qemu_arch: x86_64
qemu_version: 5.2.0
qemu_nic: virtio-net-pci
qemu_options: -machine type=pc,accel=kvm -serial mon:stdio -nographic -smbios type=1,product=VM-VEX -cpu IvyBridge,ibpb=on,spec-ctrl=on,ssbd=on,virt-ssbd=on,svm=on,erms=off
...
EOF
```

Run the following command for the system (and the UI) to know your template changes:

```
/opt/unetlab/wrappers/unl_wrapper -a fixpermissions
```

Optional: Attach Customized Junos OS Configuration

vJunos-switch similar to Juniper MX Series router, comes with almost no default Junos OS configuration. Hence, to start a new VM easily, use at least the minimal Junos OS configuration as shown below.

The following example creates a virtual configuration image with a predefined Junos OS configuration:

- Enables SSH access for the root account using the password “ABC123”.
- Enables fxp0 to get a DHCP lease for allowing SSH access to the image.
- Sets an external global name-server.
- Enables LLDP to view each link on other nodes in your network.

```
cat <<EOF >juniper.conf
system {
    host-name vjunos;
    root-authentication {
        encrypted-password "\$6\$DOvFAxW9\
$HpxgOaGEe5L6MtDJqbWepS5NT6EW23rCuu69gwwGVFr7BpzY2MHS34mPrR0LKRqoGI19tRgpz3vFJkEueW9mQ1"; ##
SECRET-DATA
    }
    services {
```

```
        ssh {
            root-login allow;
            protocol-version v2;
        }
    }
    name-server {
        8.8.8.8;
        9.9.9.9;
    }
    arp {
        aging-timer 5;
    }
    syslog {
        file interactive-commands {
            interactive-commands any;
        }
        file messages {
            any notice;
            authorization info;
        }
    }
}
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                dhcp force-discover;
            }
        }
    }
}
protocols {
    lldp {
        interface all;
    }
    lldp-med {
        interface all;
    }
}
EOF
```

The default vJunos-switch image activates only the first 10 ge-0/0/x interfaces, which is enough. But with Junos OS Release 23.2 or later, you can activate maximum of 96 interfaces. So, you can add commands as shown below to have that higher range and to edit the EVE-NG templates.

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

NOTE: If you enable more than 20 interfaces using this method, you cannot use the resulting vJunos-switch to build campus fabric configurations in the Mist GUI.

In the next step, you can use the original bash script `make-config.sh` from the vJunos-switch support site. The script creates an IDE-HD image to load your custom configuration.

You can download the image through a link. For example: https://webdownload.juniper.net/swdl/dl/anon/site/1/record/168885.html

As the script does not change with every vJunos-switch release, you can create the script using the below steps using a base64 encoded gzip file.

```
cat <<EOF >make-config.sh.gz.uue
H4sICG8BK2UAA21ha2UtY29uZmlnLnNoAI1T0W7TMBR9nr/iLO1DKy1pG97oOjRtMBWtHYKxF4So
lzitaZ1ktpPC0P4dO3azMopEnqLre88599x7O8eDe54P7qlakQ7pQNA1C5Miz/gy8qGLovwp+XKl
0Uv6iIfxqxO8r3JeMok509tCrtUJpnkSmeTzzQZNsoJkismapZFDkYxqBgoHDsE0TblaI5OFMGFV
leWGsxTfHXRk86ALUK1pYpQ0/6gNc6FwNwPPlaZ5wgx6peiS9fr4RY5YsioQfLYBvMaLdnDqwX3o
DKfuJ7RKzgJT/oNrDMfkiSQbRvOqxD7shY3xfImqjKLIpFeiqHKNMEP4gO5sfns5/UiONoVi2pSG
KbrXNzcfLt/ekSMpEMqsRvfT7fnVdH61F/F1T2RH+i2jfGOscNw+ON6riL3SkS3iGb6g28HxBDG+
jqFXLDfSrAVjknFCiKecLMRaM9EIC0sMaioHWpQL4gT881msUy5b4QNnGUlMopEy+uvhgYkq5GKJ
xI3cuCPp1ojGaEa8H5NFa1KoVsXWJnXjhe/lje1l+NzLn7Y0TYl1pqI6oxpWRZgjqIUIU6pp8Gz6
YTQ3y3fOYrNUWSGFwWmWcVc6hvXXjDo4zO7GrtEo2BW1kzzIC/+9pHdY7Tn8jwIrwW+kuU27kJl5
6Km+xfP35cDigPSStB2RkUIlkvox22kdWNv8gejlI6L+bglbhn04P9M0IO5SCPkNKH2wR0EEAAA=
EOF
base64 -d make-config.sh.gz.uue make-config.sh.gz
```

```
gunzip make-config.sh.gz
cat make-config.sh
```

NOTE: If you plan to onboard vJunos-switch to a Mist Cloud: You can directly connect VM to the Mist Cloud. Then, use the embedded device adoption commands to add the VM to the inventory. Review chapter "Default Junos OS Configuration for vJunos-switch" on page 20 and then run the following example there before you create the final configuration disk.

Use the following two CLI commands to create an image called `hdb.qcow2 in` your custom configuration.

```
chmod 777 make-config.sh ; ./make-config.sh juniper.conf hdb.qcow2
'juniper.conf' -> '/var/tmp/tmp.TYloe3JQtd/config/juniper.conf'
Formatting 'hdb.qcow2', fmt=raw size=1048576
mkfs.fat 4.1 (2017-01-24)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
/dev/loop4 has 64 heads and 32 sectors per track,
hidden sectors 0x0000;
logical sector size is 512,
using 0xf8 media descriptor, with 2048 sectors;
drive number 0x80;
filesystem has 2 12-bit FATs and 4 sectors per cluster.
FAT size is 2 sectors, and provides 502 clusters.
There is 1 reserved sector.
Root directory contains 512 slots and uses 32 sectors.
Volume ID is f9fd5527, volume label vmm-data    .
Copying file(s) to config disk hdb.qcow2
./
./config/
./config/juniper.conf
Cleaning up...
removed '/var/tmp/tmp.TYloe3JQtd/config/juniper.conf'
removed directory '/var/tmp/tmp.TYloe3JQtd/config'
removed directory '/var/tmp/tmp.TYloe3JQtd'
removed directory '/var/tmp/tmp.SbTNcpocEl'
Config disk hdb.qcow2 created
ls -l *.qcow2
-rw-r--r-- 1 root root 4011065344 Oct 11 00:08 hda.qcow2
-rw-r--r-- 1 root root    1048576 Oct 14 21:20 hdb.qcow2
```

```
# again execute the permissions fix
/opt/unetlab/wrappers/unl_wrapper -a fixpermissions
```

Ubuntu KVM and libvirtd Server BMS Environment

IN THIS SECTION

The instructions and recommendations are valid for an **Ubuntu 20.04.x** installation and might also work for Ubuntu 22.04.x. It is known that for LACP bridge support, Ubuntu 16.04 does not support the tweaks used in chapter "Linux Bridge and VM Interface Post VM Creation Changes" on page 37.

Preparations Before You Deploy

Optional: Install KVM/Qemu if You Have Not Done It Yet

The following commands show how you install the KVM hypervisor and run the required checks. Ensure you are using root directory.

```
sudo -i
cd /root
lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
```

```
Release:        20.04
Codename:       focal
uname -a
Linux aide-glb-srv-2 5.4.0-88-generic #99-Ubuntu SMP Thu Sep 23 17:29:00 UTC 2021 x86_64 x86_64
x86_64 GNU/Linux
# MANDATORY CHECK FOR NEEDED CPU FLAGS
grep 'flags' /proc/cpuinfo | head -n1 | grep -E 'vmx|svm'
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64
monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb
cat_l3 cdp_l3 invpcid_single pti intel_ppin ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority
ept vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a rdseed
adx smap intel_pt xsaveopt cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln
pts md_clear flush_l1d
apt-get update
apt-get -y upgrade
apt-get install -y qemu-kvm libvirt-daemon-system libvirt-clients net-tools bridge-utils
virtinst virt-top genisoimage
usermod -aG libvirt $USER
usermod -aG kvm $USER
#chown libvirt-qemu /var/lib/libvirt/images
# MANDATORY CHECK
kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

Optional: Create br0-bridge with DHCP Server

Once installed, the hypervisor automatically creates the default `virbr0` Linux bridge in the 192.168.122.0/24 prefix. This bridge will source NAT the remaining network interfaces and manage the DHCP server leases. You can connect the vJunos-switch VM fxp0 interface for OOB management. A limitation of this default virbr0 Linux bridge is that the DHCP lease handout is random and unpredictable.

For your lab server, we recommend you create an additional `br0` Linux bridge. This Linux bridge must NAT external interfaces to copy the same using the 192.168.10.0/24 prefix.

```
cat <<EOF >br0network.xml
<network>
  <name>br0network</name>
  <forward mode='nat'/>
```

```
  <bridge name='br0' stp='off' delay='0'/>
  <ip address='192.168.10.1' netmask='255.255.255.0'/>
</network>
EOF
virsh net-destroy br0network
virsh net-undefine br0network
virsh net-define --file /root/br0network.xml
virsh net-autostart br0network
virsh net-start br0network
virsh net-list
 Name         State    Autostart   Persistent
--------------------------------------------
 br0network   active   yes         yes
 default      active   yes         yes
brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.5254001a8a15       no              br0-nic
virbr0          8000.52540018df7d       yes             virbr0-nic
```

You've assigned a fixed IP to the vEX switches on fxp0 using DHCP, allowing you to SSH to the switches. The standard virbr0-bridge assigns a random IP address from the range 192.168.122.0/24. Instead, you create a simple DHCP server to provide static leases to known MAC addresses on br0 Linux bridge in the range 192.168.10.0/24. Then, start the vJunos-switch VM using one of the MAC addresses on the first interface to assign a predefined IP address to the fxp0 interface.

```
cat <<EOF > /root/br0-dnsmasq.conf
strict-order
user=root
pid-file=/tmp/br0-dnsmasq.pid
except-interface=lo
bind-dynamic
interface=br0
dhcp-range=192.168.10.150,192.168.10.250,255.255.255.0
dhcp-no-override
dhcp-authoritative
dhcp-option=3,192.168.10.1
dhcp-option=6,8.8.8.8
dhcp-host=52:54:00:6c:3c:00,aos-server,192.168.10.200
dhcp-host=52:54:00:6c:3c:01,sw-1,192.168.10.201
dhcp-host=52:54:00:6c:3c:02,sw-2,192.168.10.202
dhcp-host=52:54:00:6c:3c:03,sw-3,192.168.10.203
dhcp-host=52:54:00:6c:3c:04,sw-4,192.168.10.204
```

```
dhcp-host=52:54:00:6c:3c:05,sw-5,192.168.10.205
dhcp-host=52:54:00:6c:3c:06,sw-6,192.168.10.206
dhcp-host=52:54:00:6c:3c:07,sw-7,192.168.10.207
dhcp-host=52:54:00:6c:3c:08,sw-8,192.168.10.208
dhcp-host=52:54:00:6c:3c:09,sw-9,192.168.10.209
dhcp-host=52:54:00:6c:3c:0a,sw-10,192.168.10.210
dhcp-host=52:54:00:6c:3c:0b,sw-11,192.168.10.211
dhcp-host=52:54:00:6c:3c:0c,sw-12,192.168.10.212
dhcp-host=52:54:00:6c:3c:0d,sw-13,192.168.10.213
dhcp-host=52:54:00:6c:3c:0e,sw-14,192.168.10.214
dhcp-host=52:54:00:6c:3c:0f,sw-15,192.168.10.215
dhcp-host=52:54:00:6c:3c:10,sw-16,192.168.10.216
dhcp-host=52:54:00:6c:3c:99,wan-router,192.168.10.99
EOF
dnsmasq --conf-file=/root/br0-dnsmasq.conf
```

Create Linux Bridges to Simulate Network Links Between VMs

Use the default `virbr0` or the new `br0` Linux bridge when you connect the fxp0 interface of a vJunos-switch VM. As all other links need new bridges, create virtual links between devices. In this example, the bridge name tries to copy the Junos OS interface naming conventions.

```
ip link add name ge000 type bridge
ip link set ge000 up
ip link add name ge001 type bridge
ip link set ge001 up
ip link add name ge002 type bridge
ip link set ge002 up
ip link add name ge003 type bridge
ip link set ge003 up
# OPTIONAL: check your Linux-bridges
brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.525400938f81       no              br0-nic
                                                        vnet0
ge000           8000.000000000000       no
ge001           8000.000000000000       no
ge002           8000.000000000000       no
ge003           8000.000000000000       no
virbr0          8000.525400ffc947       yes             virbr0-nic
```

Default Junos OS Configuration for vJunos-switch

You might need to include a custom Junos OS configuration executed at the starting the vJunos-switch VM for the following main reasons:

- Create a root/supervisor account with a known password for a management system such as, Juniper Apstra to login. This avoids the need for a local DHCP server to push the customized configuration and supports SSH login for better debugging.
- Apply an adopt configuration for the Mist Cloud so the new switch appears in the inventory.
- Load an existing valid configuration from a previous lab at the starting of a new lab.

Customized Factory-Default

Here is an example of minimal Junos OS configuration that you load when creating a file `juniper.conf` with the following configurations:

- Username=`root`
- Password=`ABC123`
- Expected to get a DHCP lease from fxp.0
- Enable LLDP on all interfaces

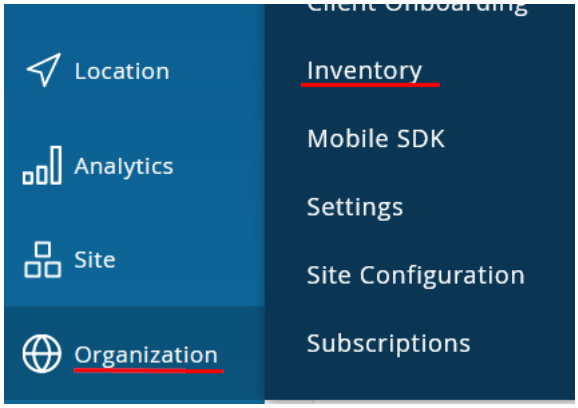
```
cat <<EOF >juniper.conf
system {
    host-name spine1;
    root-authentication {
        encrypted-password "\$6\$DOvFAxW9\
$HpxgOaGEe5L6MtDJqbWepS5NT6EW23rCuu69gwwGVFr7BpzY2MHS34mPrR0LKRqoGI19tRgpz3vFJkEueW9mQ1"; ##
SECRET-DATA
    }
    services {
        ssh {
            root-login allow;
            protocol-version v2;
        }
    }
    name-server {
        8.8.8.8;
        9.9.9.9;
    }
    arp {
```

```
        aging-timer 5;
    }
    syslog {
        file interactive-commands {
            interactive-commands any;
        }
        file messages {
            any notice;
            authorization info;
        }
    }
}
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                dhcp force-discover;
            }
        }
    }
}
protocols {
    lldp {
        interface all;
    }
    lldp-med {
        interface all;
    }
}
EOF
```

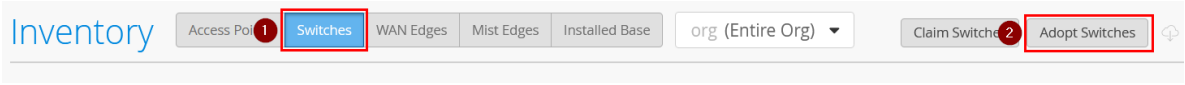
Optional: Adding Junos OS Configuration for Adoption to Mist Cloud

In this step, you'll add Junos OS configuration to make the switch appear automatically in the Mist Cloud.

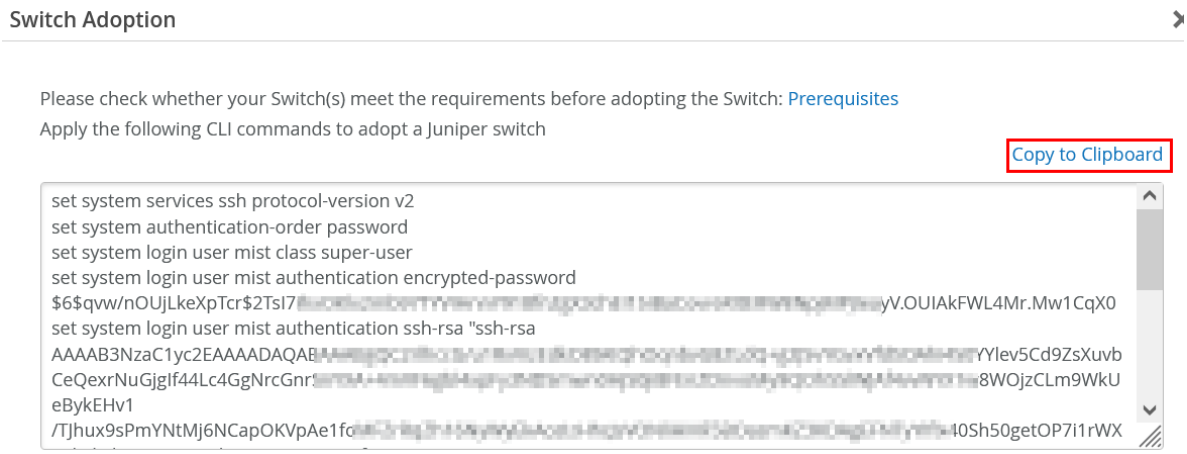
1. Go to **Organization -> Inventory.**



2. Select **Switches** and then click **Adopt Switches**.



3. Click **Copy to Clipboard**.



4. Open a file `adopt-template.txt` from your preferred editor and paste the gathered information in this file. Then, save and close the file.

```
vi adopt-template.txt
```

NOTE: You can use the same Mist Cloud adoption code to onboard all switches in the same organization.

5. Convert the information using a template to change it in a different Junos OS configuration format. First, you must extract five variables from the file `adopt-template.txt` using bash shell commands.

```
MISTVAR1=`cat adopt-template.txt | sed 's/\"//g' | grep "set system login user mist
authentication encrypted-password" | awk '{print $8}'`
MISTVAR2=`cat adopt-template.txt | sed 's/\"//g' | grep "set system login user mist
authentication ssh-rsa" | awk '{print $8 " " $9}'`
MISTVAR3=`cat adopt-template.txt | sed 's/\"//g' | grep "set system services outbound-ssh
client mist secret" | awk '{print $8}'`
MISTVAR4=`cat adopt-template.txt | sed 's/\"//g' | grep "set system services outbound-ssh
client mist" | grep "port" | awk '{print $7}'`
MISTVAR5=`cat adopt-template.txt | sed 's/\"//g' | grep "set system services outbound-ssh
client mist" | grep "device-id" | awk '{print $8}'`
```

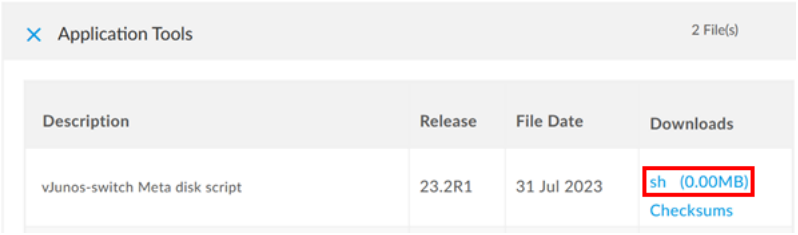
In this step, you add back the Junos OS configuration, which does not depend on set commands.

```
cat <<EOF >>juniper.conf
system {
    login {
        user mist {
            class super-user;
            authentication {
                encrypted-password "$MISTVAR1";
                ssh-rsa "$MISTVAR2";
            }
        }
    }
    services {
        outbound-ssh {
            client mist {
                device-id "$MISTVAR5";
                secret "$MISTVAR3";
                keep-alive {
                    retry 12;
                    timeout 5;
                }
                services netconf;
                $MISTVAR4 {
                    port 2200;
                    retry 1000;
                    timeout 60;
```

```
                }
            }
        }
    }
}
EOF
# review the final config
cat juniper.conf
```

Creating Virtual Disk with Junos OS Configuration for VM

1. Create an HD image to load your custom configuration using the original bash script `make-config.sh` from the vJunos-switch support site.



2. Download the image through a link. For example: https://webdownload.juniper.net/swdl/dl/anon/site/1/record/168885.html.

   If you can't find an image, use the copy below.

```
#!/bin/bash
#
# make-config.sh
#
# Copyright (c) 2023, Juniper Networks, Inc.
# All rights reserved.
#
# Create a config metadisk from a supplied juniper.conf to attach
# to a vJunos VM instance
#
usage() {
        echo "Usage :  make-config.sh <juniper-config> <config-disk>"
        exit 0;
}
cleanup () {
        echo "Cleaning up..."
        umount -f -q $MNTDIR
        losetup -d $LOOPDEV
```

```
        rm -rfv $STAGING
        rm -rfv $MNTDIR
}
cleanup_failed () {
        cleanup;
        rm -rfv $2
        exit 1
}
if [ $# != 2 ]; then
        usage;
fi
STAGING=`mktemp -d -p /var/tmp`
MNTDIR=`mktemp -d -p /var/tmp`
mkdir $STAGING/config
cp -v $1 $STAGING/config
qemu-img create -f raw $2 1M
LOOPDEV=`losetup --show -f $2`
if [ $? != 0 ]; then
        cleanup_failed;
fi
mkfs.vfat  -v -n "vmm-data" $LOOPDEV
if [ $? != 0 ]; then
        echo "Failed to format disk $LOOPDEV; exiting"
        cleanup_failed;
fi
mount -t vfat $LOOPDEV $MNTDIR
if [ $? != 0 ]; then
        echo "Failed to mount metadisk $LOOPDEV; exiting"
        cleanup_failed;
fi
echo "Copying file(s) to config disk $2"
(cd $STAGING; tar cvzf $MNTDIR/vmm-config.tgz .)
cleanup
echo "Config disk $2 created"
exit 0
```

3. Create a qcow2-Image that includes your customized configuration using `make-config.sh`.

```
./make-config.sh juniper.conf myconfig.qcow2
'juniper.conf' -> '/var/tmp/tmp.S9uwgranof/config/juniper.conf'
Formatting 'myconfig.qcow2', fmt=raw size=1048576
```

```
mkfs.fat 4.1 (2017-01-24)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
/dev/loop7 has 64 heads and 32 sectors per track,
hidden sectors 0x0000;
logical sector size is 512,
using 0xf8 media descriptor, with 2048 sectors;
drive number 0x80;
filesystem has 2 12-bit FATs and 4 sectors per cluster.
FAT size is 2 sectors, and provides 502 clusters.
There is 1 reserved sector.
Root directory contains 512 slots and uses 32 sectors.
Volume ID is 9136e65a, volume label vmm-data    .
Copying file(s) to config disk myconfig.qcow2
./
./config/
./config/juniper.conf
Cleaning up...
removed '/var/tmp/tmp.S9uwgranof/config/juniper.conf'
removed directory '/var/tmp/tmp.S9uwgranof/config'
removed directory '/var/tmp/tmp.S9uwgranof'
removed directory '/var/tmp/tmp.iz7I2RJJAI'
Config disk myconfig.qcow2 created
ls -l myconfig.qcow2
-rw-r--r-- 1 root root 1048576 Apr 19 19:46 myconfig.qcow2
```

4. Copy your configuration image to the final destination for KVM VMs.

```
cp myconfig.qcow2 /var/lib/libvirt/images/spine1-config.qcow2
```

5. Insert the bold line as shown below to the VM virt-install configuration when you create a new vJunos-switch VM. An example is described in chapter "Proxmox Virtual Environment" on page 43.

```
.
--disk path=/var/lib/libvirt/images/spine1.qcow2,cache=writeback,bus=virtio \
--disk path=/var/lib/libvirt/images/spine1-config.qcow2 \
--import \
.
```

Deployment of vJunos-switch VM Using virt-install CLI

For each vJunos-switch VM, you need an individual copy of the base image for the system to write to it. The configuration below uses the KVM backing file method to create an image with the changes made and reads from the original image. This method speeds up the VM start and saves storage. You might have to modify the file `/etc/libvirt/qemu.conf` to add yourself as a user and group. Then, run `sudo systemctl restart libvirtd` to use this feature.

```
qemu-img create -b vjunos-switch-23.1R1.8.qcow2 -f qcow2 /var/lib/libvirt/images/spine1.qcow2
```

Backup files must always allow you to copy the image for each VM.

```
cp vjunos-switch-23.1R1.8.qcow2 /var/lib/libvirt/images/spine1.qcow2
```

Finally, you can successfully start the vJunos-switch VM in the configuration below without changing the parameters in bold. The VM first Ethernet interface is always fxp0. After you've set the MAC address and configured the DHCP server to assign 192.168.10.201 as static DHCP lease, you can directly start a remote shell later.

```
virt-install -n spine1 -r 5120 --vcpus=4 \
--sysinfo smbios,system.product=VM-VEX \
--hvm --cpu IvyBridge,require=vmx \
--disk path=/var/lib/libvirt/images/spine1.qcow2,cache=writeback,bus=virtio \
--import \
-w bridge=br0,model=virtio,mac="52:54:00:6c:3c:01" \
-w bridge=ge000,model=virtio \
-w bridge=ge001,model=virtio \
-w bridge=ge002,model=virtio \
-w bridge=ge003,model=virtio \
--nographics --noautoconsole
Starting install...
Domain creation completed.
virsh domiflist spine1
 Interface   Type     Source   Model    MAC
-------------------------------------------------------------
 vnet1       bridge   br0      virtio   52:54:00:6c:3c:01
 vnet2       bridge   ge000    virtio   52:54:00:ac:2f:63
 vnet3       bridge   ge001    virtio   52:54:00:90:ea:79
 vnet4       bridge   ge002    virtio   52:54:00:24:0f:1d
 vnet5       bridge   ge003    virtio   52:54:00:11:45:56
```

```
brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.525400938f81       no              br0-nic
                                                        vnet0
                                                        vnet1
brtest          8000.000000000000       no
ge000           8000.fe5400ac2f63       no              vnet2
ge001           8000.fe540090ea79       no              vnet3
ge002           8000.fe5400240f1d       no              vnet4
ge003           8000.fe5400114556       no              vnet5
virbr0          8000.525400ffc947       yes             virbr0-nic
# OPTIONAL: check the xml config create with the official documentation
virsh dumpxml spine1
# OPTIONAL: open a console to the VM to see what it does
virsh console --force spine1
```

Deployment of vJunos-switch VM Using virt-manager GUI

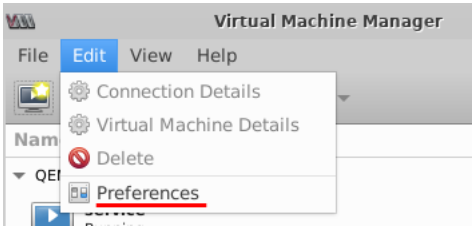
For installing vJunos-switch through virt-manager GUI, you must manually change few XML based configuration files. So, the GUI does not provide any benefit for this VM.

1. Obtain a copy of the base VM to the libvirtd image directory as shown.

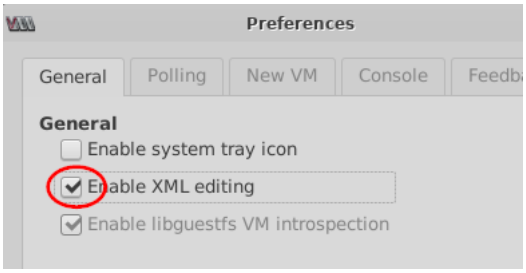
```
cp vjunos-switch-23.1R1.2.qcow2 /var/lib/libvirt/images/spine1.qcow2
```

```
root@megalon6:~#
root@megalon6:~# cp vjunos-switch-23.1R1.2.qcow2 /var/lib/libvirt/images/spine1.qcow2
root@megalon6:~#
```

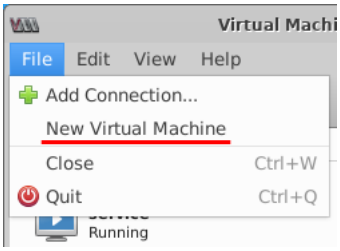
2. Click **Edit -> Preferences** to enable XML editing for this VM.



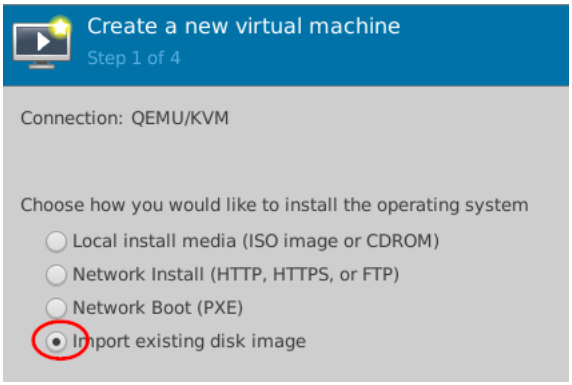
3. Select **Enable XML editing**.



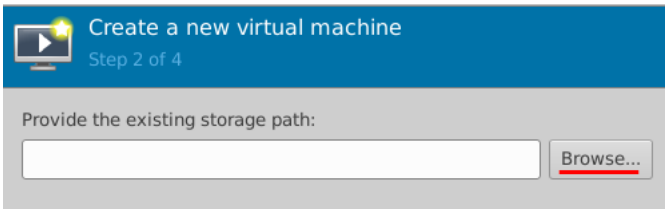
4. Select **File -> New Virtual Machine**.



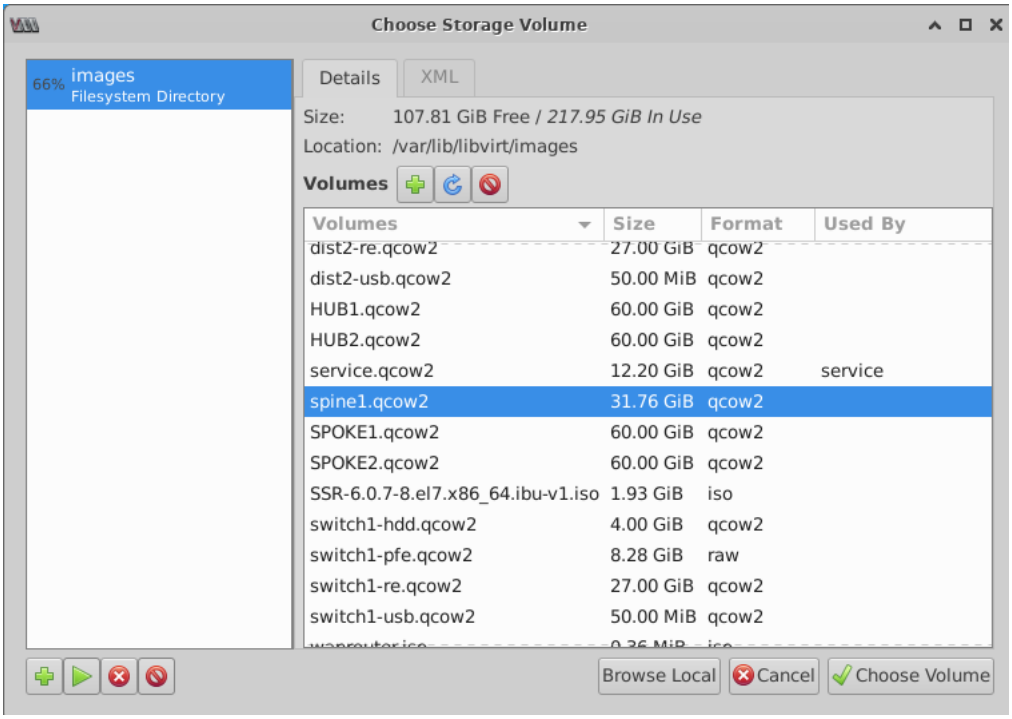
5. Select **Import existing disk image** for installation.



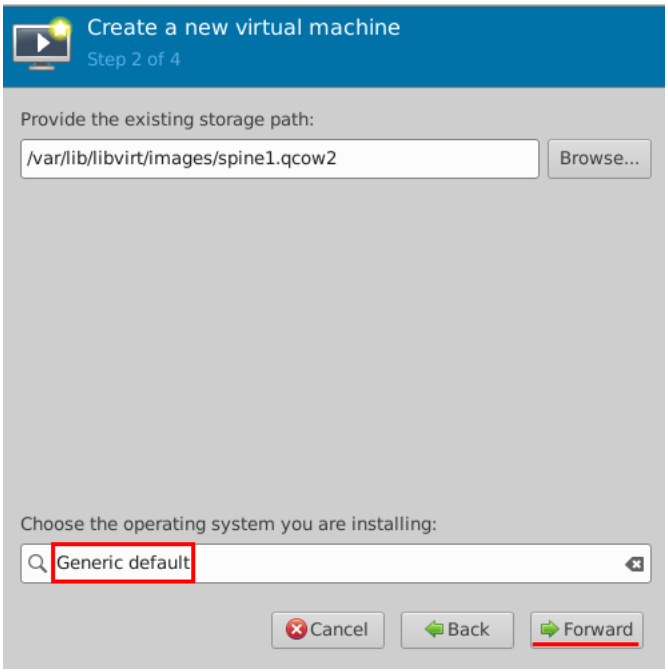
6. Click **Browse**.



7. Select the image that you have copied and click **Choose Volume**.

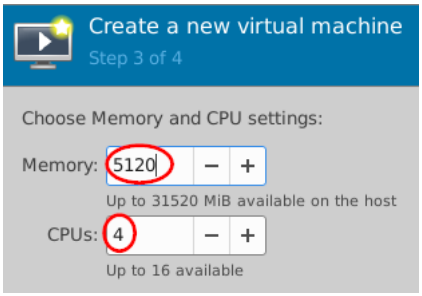


8. Select **Generic default** for OS and click **Forward**.



9. Click + to configure:

- Memory=5120
- CPUs=4



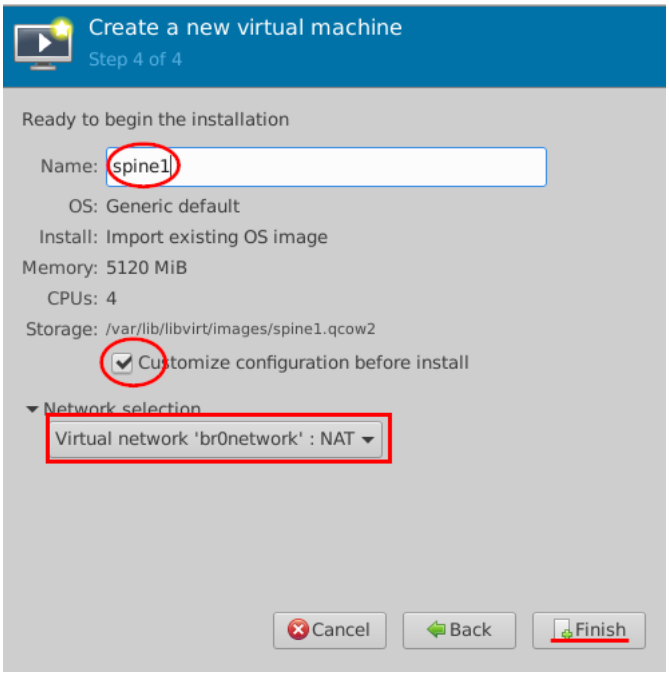
10. Then, configure:

    a. Name=`spine1`

    b. Select **Customize configuration before install**.

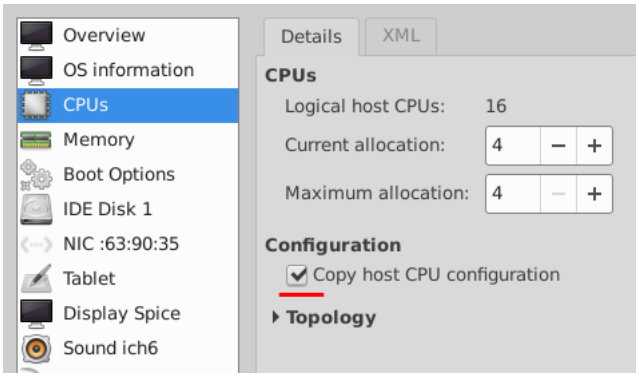
    c. Select **Virtual network 'br0network': NAT** for now. Network selection for the first interface (fxp0) must be a NAT network.

    d. Click **Finish**.

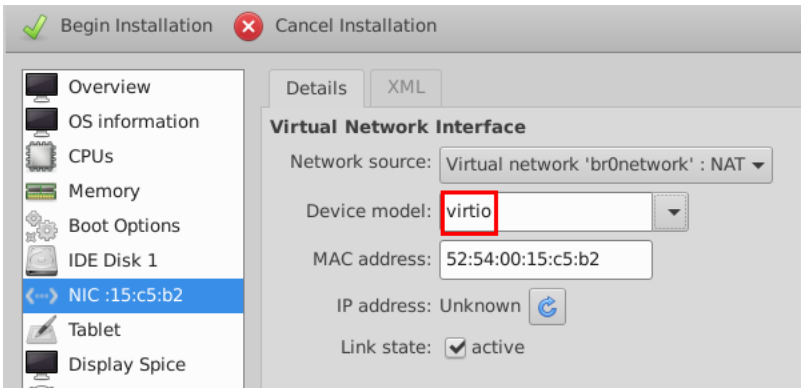


11. Click **CPUs**.

    The host CPU is configured. You can click **XML** to change the configuration later as the other options are not required.

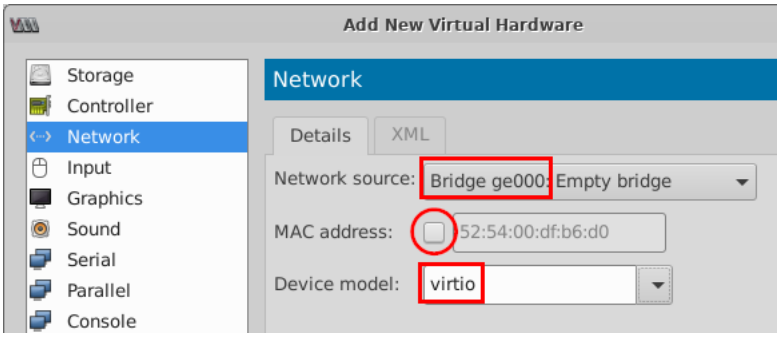


12. Change the Device model as **virtio** for the first interface.

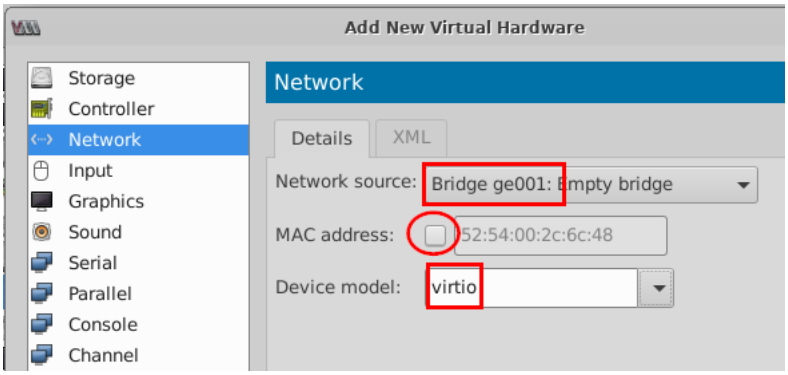


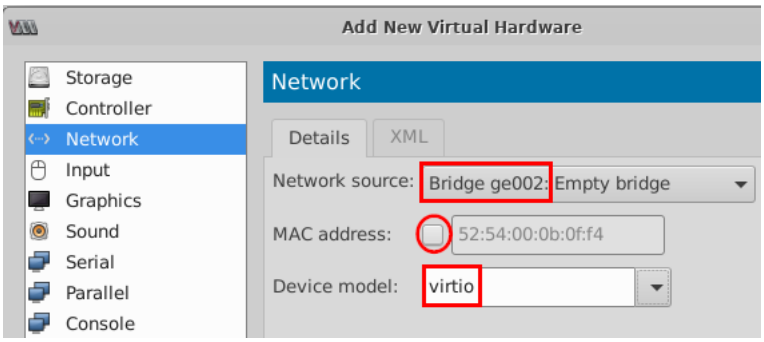
13. Add a new network interface using the add function for interfaces and configure:

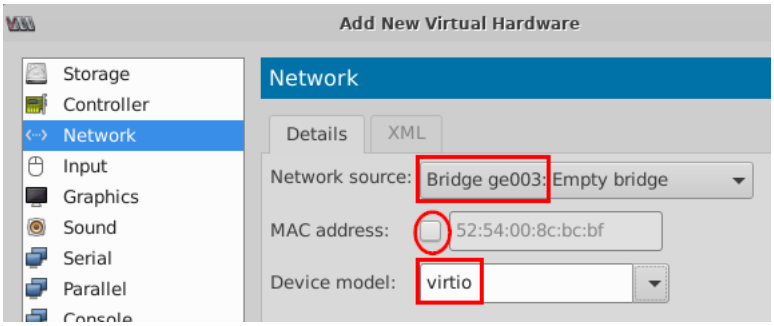
- Network source=`Bridge ge000`
- Uncheck=`MAC address`
- Device model=`virtio`



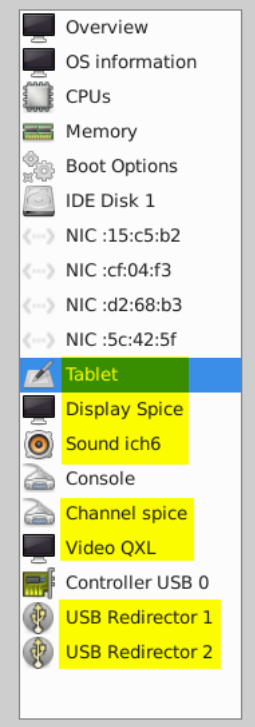
14. Repeat Step 13 to add network source `Bridge ge001`, `Bridge ge002` and `Bridge ge003`.



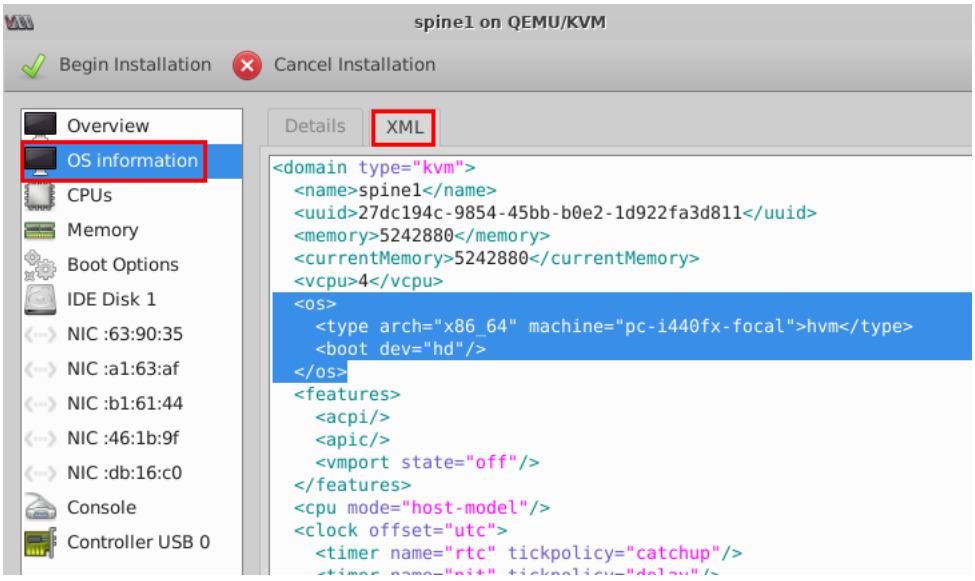




15. Optionally, you can remove the default interface and options highlighted in yellow in the figure below as they are not required for vJunos-switch VM.



16. Select **OS information** and then **XML** for editing. Delete everything between <os> and </os> and replace it with the highlighted lines as show below. The replaced configuration instructs special BIOS command to ensure that the VM acts as a vEX9214 and not as a vMX.

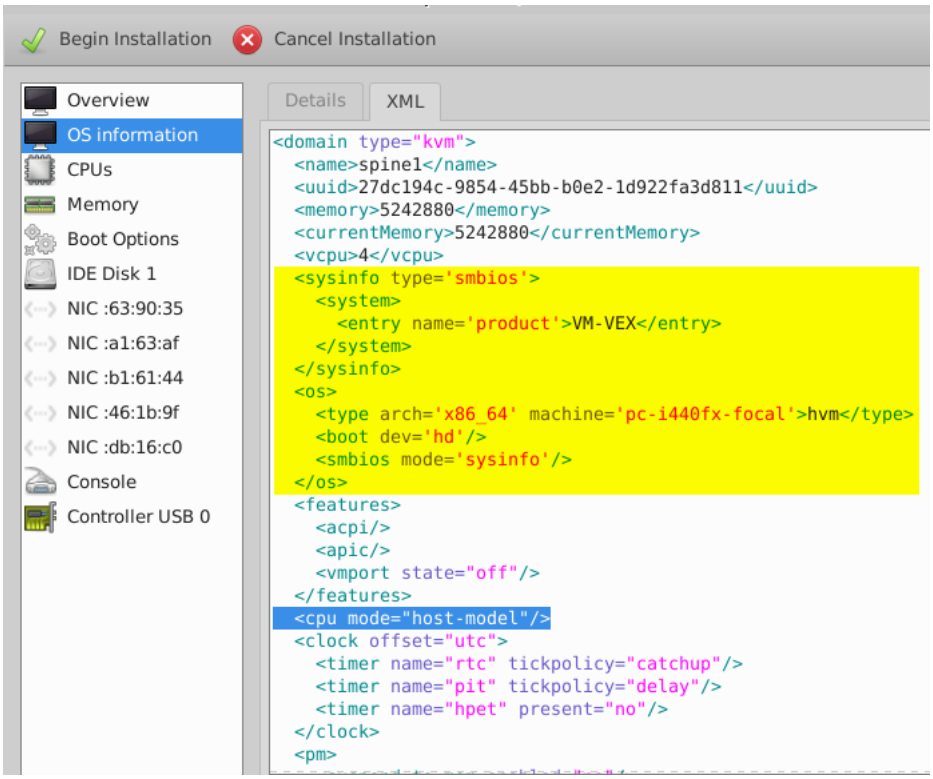


a. Insert the following configuration as shown below:

```
<sysinfo type='smbios'>
  <system>
```

```
    <entry name='product'>VM-VEX</entry>
  </system>
</sysinfo>
<os>
  <type arch='x86_64' machine='pc-i440fx-focal'>hvm</type>
  <boot dev='hd'/>
  <smbios mode='sysinfo'/>
</os>
```

The replaced configuration is shown as highlighted in yellow in the figure below.

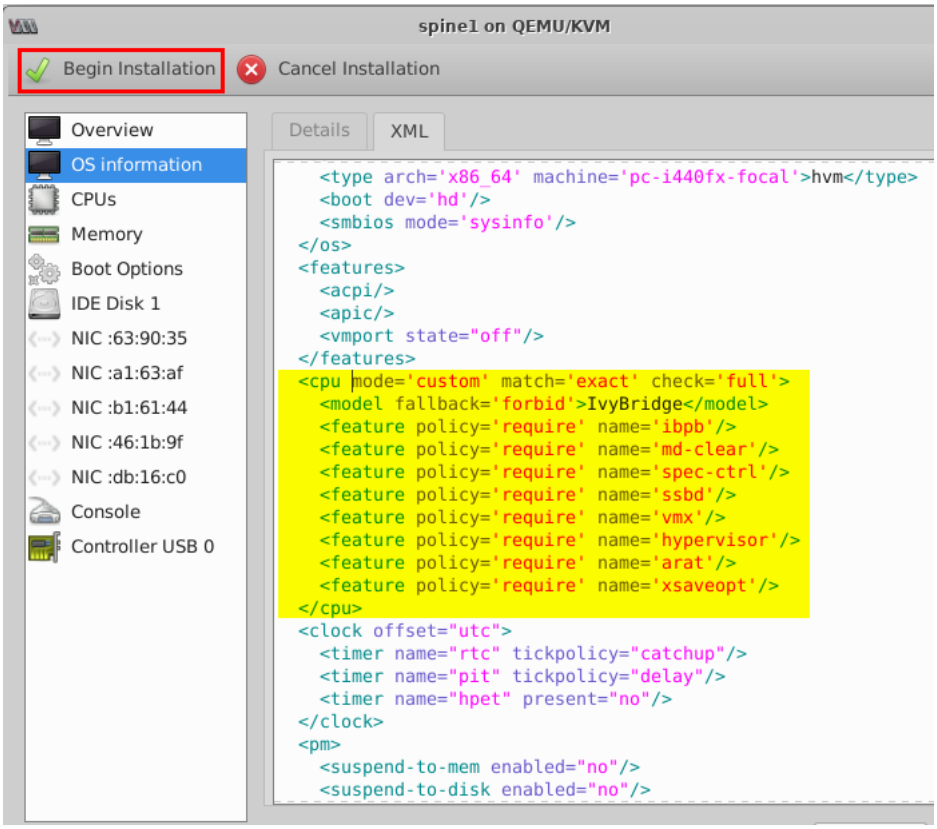


b. Insert the following configuration from <cpu as shown in the figure above.

```
<cpu mode='custom' match='exact' check='full'>
  <model fallback='forbid'>IvyBridge</model>
  <feature policy='require' name='ibpb'/>
  <feature policy='require' name='md-clear'/>
  <feature policy='require' name='spec-ctrl'/>
  <feature policy='require' name='ssbd'/>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='hypervisor'/>
  <feature policy='require' name='arat'/>
```

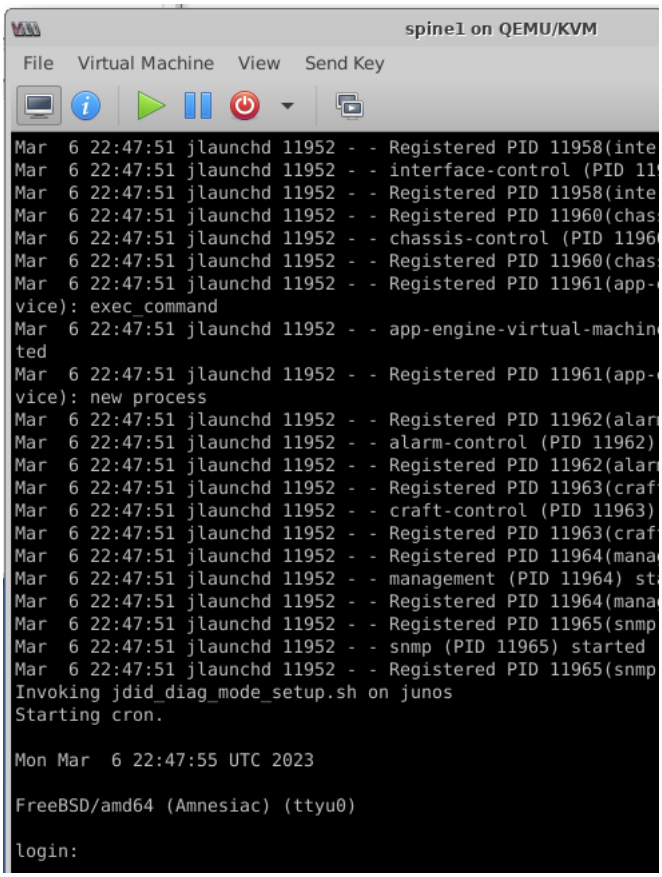
```
    <feature policy='require' name='xsaveopt'/>
  </cpu>
```

The replaced configuration is shown as highlighted in yellow in the figure below.



17. Click **apply** and **Begin Installation**.

The installation process might take three minutes and you'll be prompted with a VM login.



Linux Bridge and VM Interface Post VM Creation Changes

After the VM is launched, you must change the interface and Linux bridge configuration. The bridges that you created with the default configuration in chapter "Preparations Before You Deploy" on page 16, are applicable for an EVPN VXLAN fabric.

The default Linux bridge does not support:

- **LLDP** message transport, which means you cannot see any link neighbors.
- **LACP 802.3ad** message transport, which prevents you from building LAG.
- **Large MTUs**, which causes fragmentation of VXLAN messages. Using VXLAN, the fabric must have MTUs at least 50 bytes larger than the attached clients. Image the attached Desktop VM and all fabric links with same default MTU of 1500, where the transport links always add extra 50 bytes. The small packets such as, ICMP Ping works well but not when the clients use the full MTU of 1500 bytes.
- **802.1X** wired client authentication is blocked.

Hence, you need to change the virtual interface and Linux bridges after creating the VM to support these features. This change ensures that you build a virtual EVPN VXLAN fabric smoothly.

1. Create a bash script using the following commands to perform all the required functions.

```
rm -f vm-bridge-update.sh
touch vm-bridge-update.sh
chmod 777 vm-bridge-update.sh
vi vm-bridge-update.sh
```

2. Copy and paste the below configuration to your editor. Then, save and close.

```
#!/bin/bash
virsh domiflist $1 | tail -n +4  > /tmp/vmbridgelist.txt
sed -i '/^$/d' /tmp/vmbridgelist.txt
# cat /tmp/vmbridgelist.txt
while IFS= read -r line
do
  INTERFACE=`echo $line | awk '{ print $1 }'`
  NTYPE=`echo $line | awk '{ print $2 }'`
  BRIDGE=`echo $line | awk '{ print $3 }'`
  if [ "$NTYPE" == "bridge" ]; then
    # change MTU to higher value
    RUNME="ip link set dev "$INTERFACE" mtu 9200"
    echo $RUNME
    eval $RUNME
    # enable LLDP and 802.1x on bridge
    RUNME="echo 65528 > /sys/class/net/"$BRIDGE"/bridge/group_fwd_mask"
    echo $RUNME
    eval $RUNME
    # enable LACP on link
    RUNME="echo 16388 > /sys/class/net/"$INTERFACE"/brport/group_fwd_mask"
    echo $RUNME
    eval $RUNME
  fi
done < /tmp/vmbridgelist.txt
num=0
while IFS= read -r line
do
  INTERFACE=`echo $line | awk '{ print $1 }'`
  NTYPE=`echo $line | awk '{ print $2 }'`
```

```
  BRIDGE=`echo $line | awk '{ print $3 }'`
  if [ "$NTYPE" == "bridge" ]; then
    MTU=`cat /sys/class/net/$BRIDGE/mtu`
    if [ "$MTU" != "9200" ]; then
      echo 'Warning! Bridge:'$BRIDGE' did not follow new MTU setting of
interface:'$INTERFACE' check other interfaces attached to same bridge and correct please!'
      num=1
    fi
  fi
done < /tmp/vmbridgelist.txt
exit $num
```

3. Run this script on a demonstration VM, where you can monitor the commands that are required to apply the changes.

```
./vm-bridge-update.sh core1
ip link set dev vnet4 mtu 9200
echo 65528 > /sys/class/net/exfabric1/bridge/group_fwd_mask
echo 16388 > /sys/class/net/vnet4/brport/group_fwd_mask
ip link set dev vnet5 mtu 9200
echo 65528 > /sys/class/net/uplink1/bridge/group_fwd_mask
echo 16388 > /sys/class/net/vnet5/brport/group_fwd_mask
ip link set dev vnet6 mtu 9200
echo 65528 > /sys/class/net/fabric1/bridge/group_fwd_mask
echo 16388 > /sys/class/net/vnet6/brport/group_fwd_mask
ip link set dev vnet7 mtu 9200
echo 65528 > /sys/class/net/fabric2/bridge/group_fwd_mask
echo 16388 > /sys/class/net/vnet7/brport/group_fwd_mask
Warning! Bridge:uplink1 did not follow new MTU setting of interface:vnet5 check other
interfaces attached to same bridge and correct please!
```

You've intentionally given a warning about the non-upgraded MTU on a certain interface. This warning implies that a Linux bridge always has the lowest MTU of any attached network interface. Hence, ensure to upgrade all interfaces attached to a single bridge. Else, your MTU changes are not implemented.

NOTE: The script intentionally does not upgrade the first interface of a VM, which is usually fxp0. If you want to include this interface, change the second line to "virsh domiflist $1 | tail -n **+3** > /tmp/vmbridgelist.txt".

NOTE: Tweaking Linux bridge works only with Ubuntu 20.04 or higher.

Optional: Optimizing Your KVM Server

Ultra Kernel Samepage Merging Kernel

Using a kernel that supports Ultra Kernel Samepage Merging can save 30 - 40% memory compared to the standard KSM support when launching multiple vJunos-switch VMs. This is beneficial for a lab and systems such as EVE-NG which includes this Kernel by default. If the system runs on Ubuntu 20.04.x, which does not support higher kernels, you can use the following configurations to build such a kernel from the official GitHub repository.

NOTE: Ensure that you have disabled Secure Boot in the BIOS. If not, the unsigned kernel does not load.

```
lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal

uname -r
5.4.0-135-generic

sudo apt-get install -y build-essential flex bison git libssl-dev libncurses-dev libelf-dev zstd
git clone https://github.com/dolohow/uksm.git
```

1. Enter https://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/focal/refs/ in a browser.
   a. If a normal `5.4.x` kernel is installed on your server, then run the following commands:

```
rm -rf focal
git clone --depth 1 --single-branch --branch Ubuntu-5.4.0-135.152 https://
```

```
git.launchpad.net/\~ubuntu-kernel/ubuntu/+source/linux/+git/focal
cd focal
patch -p1 < ~/uksm/v5.x/uksm-5.4.patch
```

b. If a newer kernel such as, `5.15.x` with Hardware Enablement Stack (HWE) support is installed on your server, then run the following commands:

```
rm -rf focal
git clone --depth 1 --single-branch --branch Ubuntu-hwe-5.15-5.15.0-57.63_20.04.1 https://
git.launchpad.net/\~ubuntu-kernel/ubuntu/+source/linux/+git/focal
cd focal
patch -p1 < ~/uksm/v5.x/uksm-5.15.patch
```

2. Regardless of your kernel version, proceed with the following configuration to build and install the USKM kernel.

```
make oldconfig
Enable KSM for page merging (KSM) [Y/n/?] y
  Choose UKSM/KSM strategy
  > 1. Ultra-KSM for page merging (UKSM) (NEW)
    2. Legacy KSM implementation (KSM_LEGACY) (NEW)
  choice[1-2?]: 1
# patches for this kernel to compile
scripts/config --disable DEBUG_INFO

sed -i 's/CONFIG_SYSTEM_TRUSTED_KEYS="debian\/canonical-certs.pem"/
CONFIG_SYSTEM_TRUSTED_KEYS=""/g' .config
sed -i 's/CONFIG_SYSTEM_REVOCATION_KEYS="debian\/canonical-revoked-certs.pem"/
CONFIG_SYSTEM_REVOCATION_KEYS=""/g' .config
cat .config | grep _KEYS
# The below command is slower but maybe used for debugging when building the kernel fails
# make deb-pkg LOCALVERSION=-uksm
make -j$(nproc) deb-pkg LOCALVERSION=-uksm
ls -la ../*deb
-rw-r--r-- 1 root root  11650688 Jan  7 13:11 ../linux-headers-5.4.212-uksm_5.4.212-
uksm-1_amd64.deb
-rw-r--r-- 1 root root  62563988 Jan  7 13:12 ../linux-image-5.4.212-uksm_5.4.212-
uksm-1_amd64.deb
-rw-r--r-- 1 root root   1072104 Jan  7 13:11 ../linux-libc-dev_5.4.212-uksm-1_amd64.deb
cd ..
```

```
sudo dpkg -i linux-headers-5.4.212-uksm_5.4.212-uksm-1_amd64.deb
sudo dpkg -i linux-image-5.4.212-uksm_5.4.212-uksm-1_amd64.deb
sudo update-grub
sudo reboot
```

3. After the server is up again, check if you are running on the new kernel.

```
uname -r
5.4.212-uksm
```

Turn Off Security Mitigations

> **NOTE**: Avoid these instructions if your server is in production or accessible through the Internet. You can revise the security checks to gain CPU performance in your lab. You are warned.

To avoid 20% CPU performance loss, you'll disable all mitigations against meltdown/spectre vulnerabilities. This server does not host VMs from different users. As you manage all the VMs in your lab and it's not a public offered production-grade system, you prioritize performance over security.

```
#we ARE already a VM so to have most speed turn off security patches
#add mitigations=off to your kernel parameters like below
sudo vi /etc/default/grub
.
.
GRUB_CMDLINE_LINUX_DEFAULT=""
GRUB_CMDLINE_LINUX="mitigations=off"
# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
.
.
sudo update-grub
# make latest kernel >=5.4 active through reboot of VM
sudo reboot
# you should see the below now with disabled security tweaks
grep . /sys/devices/system/cpu/vulnerabilities/* | egrep -i 'meltdown|spectre'
/sys/devices/system/cpu/vulnerabilities/meltdown:Vulnerable
/sys/devices/system/cpu/vulnerabilities/spectre_v1:Vulnerable: __user pointer sanitization and
```

```
usercopy barriers only; no swapgs barriers
/sys/devices/system/cpu/vulnerabilities/spectre_v2:Vulnerable, IBPB: disabled, STIBP: disabled
```

Proxmox Virtual Environment

IN THIS SECTION

As another option, you can consider building a lab in Proxmox VE. Internally, the hypervisor on EVE-NG, Ubuntu native KVM with libvirtd, and Proxmox VE is the same. In all three environments, QEMU runs the VM. Each environment has its own CLI and GUI and uses either Debian or Ubuntu Linux distributions.

Proxmox VE benefits against EVE-NG and Ubuntu native KVM with libvirtd are:

- Easy to build clusters of hypervisors, which limits the scope of a single BMS.
- Easy to attach shared storage such as Ceph.
- Virtualize networks amongst servers using SDN option.
- REST API operates your systems.

Disadvantages of Proxmox VE benefits against EVE-NG and Ubuntu native KVM with libvirtd are:

- Building an UKSM kernel cannot save RAM usage of multiple vJunos-switch instances. Hence, each vJunos-switch VM needs 5 GB RAM.
- Does not run compressed or backing qcow2 images, instead they are expanded as raw image on the storage option. Hence, each vJunos-switch VM needs 32 GB storage.

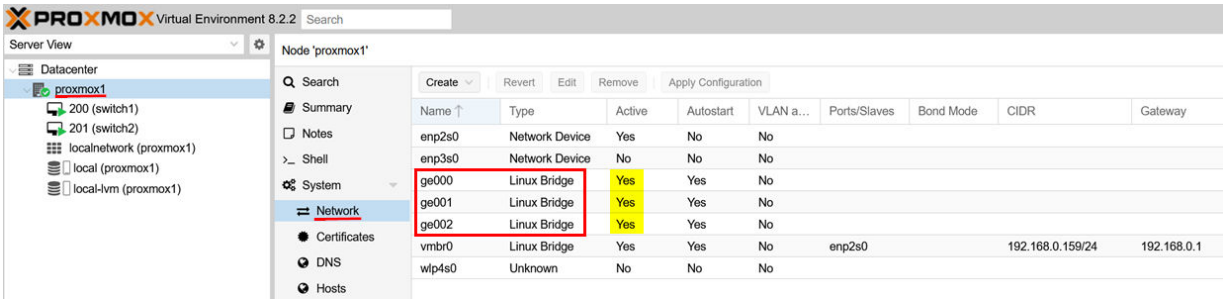
This document includes examples of creating vJunos-switch VMs on Proxmox VE with a locally configured single Proxmox server and the standard Linux bridges. This helps to compare with the previously described other two environments. As you've not used the Proxmox GUI for VM, you must run configuration changes locally after creating juniper.conf images and "Linux Bridge and VM interface

post VM creation changes on Proxmox VE" on page 52. The CLI example makes it easier for you to include it in a script to launch multiple vJunos-switch VMs.

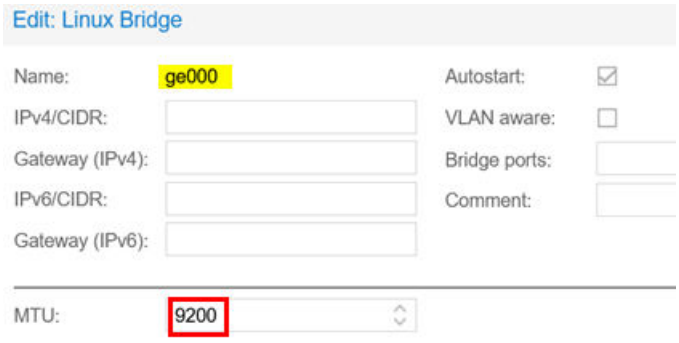
> **NOTE**: For scale out labs with multiple servers, we recommend using SDN with VXLAN as network transport option instead of local Linux bridges.

Proxmox VE Preparations

After installing the hypervisor, create the networks to use for vJunos-switch VMs and others in your lab. As in the example above, use the Proxmox GUI to create standard Linux bridges as the three shown below and ensure that they are activated.



Assign a name to each Linux bridge and you can optionally set the MTU to 9200. You can change the MTU value using the script after you create the VM. Avoid populating/changing any of the other values.



For all the remaining steps, use SSH to the server to run BASH commands locally. First, you download the qcow2-image of vJunos-switch to the server.

```
mkdir -p /root/download
```

Now, download your free copy of vJunos-switch VM to the directory using URL: https://support.juniper.net/support/downloads/?p=vjunos and then verify if the copy is downloaded.

```
ls -l /root/download/
-rw-r--r-- 1 root root 3966631936 Aug  1  2023 vJunos-switch-23.2R1.14.qcow2
```

Deploy a vJunos-switch VM on Proxmox VE

NOTE: Avoid creating the initial vJunos-switch VM using the Proxmox GUI as GUI might add additional parameters causing the VM to not to work properly. Instead, create the initial VM through CLI and set it as a template. Then, use this template to launch all further VMs from the GUI.

Using BASH, perform the next steps on the server locally:

1. Configure VM individually:
   a. The VM ID/Number. In the example, it is `200`.
   b. The storage where the image of the VM runs from. In the example, it is storage `local-lvm`.
2. Delete if an existing VM with the same ID is running. This is useful if you made an error and want to retry.
3. Create the new vJunos-switch VM with all required parameters to start it correctly later:
   a. Name of the VM. In the example, `vswitch`. You can change the name.
   b. RAM and CPU. Do not change.
   c. Special BIOS and CPU options that are required for this VM to come up correctly. Do not change the options.
   d. Boot order and serial screen. Do not change.
   e. First, network `net0` that gets assigned to the fxp0 interface of the VM. Change, if required but ensure that network can provide a DHCP lease for the VM.
   f. Second, more networks starting with `net1`, which will be the interface `ge-0/0/0` of the vJunos-switch VM. You will need to change that according to your lab design using more interfaces and

other Linux bridges. We recommend that you keep the option `firewall=0` for each of those interfaces to not overcomplicate the internal design.

4. Import the vJunos-switch qcow2-image into the selected storage option. You might need to change the vJunos-switch qcow2 image file location.
5. Import the configuration image location to extract to a BASH variable.
6. Add the image location to the created VM to boot from.
7. Create a default `juniper.conf` with our initial Junos OS configuration for this VM.
8. Use the `make-config.sh` script to create an image that embeds your individual `juniper.conf` file.
9. Import the Junos OS configuration image to the selected storage option.
10. Import the configuration image location to extract to a BASH variable.
11. Add the configuration image location to the created VM.
12. Check and review the complete configuration of the VM.
13. Optional: Use the VM as template for future launches of vJunos-switch:
    a. Define the current VM as template.
    b. Select a new VMID for the clone.
    c. Create a clone VM to use it later.
    d. Change the interface assignments for the clone if required.
14. Launch the VM or its clone.
15. Review the Linux bridge assignment locally for the started VM.
16. Review on the Proxmox GUI if the VM has started and then access the console.

```
# configure the management ID for the VM and your storage location
VMID="200"
VMSTORAGE="local-lvm"
# make sure any prior instance of this VM is down and deleted
qm stop $VMID
qm destroy $VMID
# create a new VM without a virtual disk
qm create $VMID --name switch1 --memory 5120 --cores 4 \
--args "-machine accel=kvm:tcg -smbios type=1,product=VM-VEX -cpu 'host,kvm=on'" \
--boot order=virtio0 --serial0 socket \
```

```
--net0 virtio,bridge=vmbr0 \
--net1 virtio,bridge=ge000,firewall=0 \
--net2 virtio,bridge=ge001,firewall=0 \
--net3 virtio,bridge=ge002,firewall=0
# import the vJunos image as qcow2 format in proxmox
qm disk import $VMID /root/download/vJunos-switch-23.2R1.14.qcow2 $VMSTORAGE --format qcow2
| tee diskimport.txt
.
.
transferred 31.6 GiB of 31.8 GiB (99.52%)
transferred 31.8 GiB of 31.8 GiB (100.00%)
transferred 31.8 GiB of 31.8 GiB (100.00%)
Successfully imported disk as 'unused0:local-lvm:vm-200-disk-0'
# extract image location from import
VMIMAGE=`cat diskimport.txt | grep "imported disk" | awk '{print $5}' | sed 's/.:/ /' | awk
'{print $2}' | sed 's/.$//'`
echo $VMIMAGE
local-lvm:vm-200-disk-0
# attach the qcow2 disk to the vm
qm set $VMID --virtio0 $VMIMAGE,iothread=1,size=32G
update VM 200: -virtio0 local-lvm:vm-200-disk-0,iothread=1,size=32G
```

Review the chapter "Default Junos OS Configuration for vJunos-switch" on page 20. This chapter guides you with the process of creating an individual Junos OS configuration for your vJunos-switch VM, which is similar on the other environments. This chapter also guides you to add an adopt configuration, which allows each new vJunos-switch VMs automatically appear in the Mist Cloud inventory. Here, without repeating the same steps, you used a minimal startup configuration for remote SSH access as `root` with the password `ABC123` on the fxp0 interface.

```
cat <<EOF >juniper.conf
system {
    host-name vjunos;
    root-authentication {
        encrypted-password "\$6\$DOvFAxW9\
$HpxgOaGEe5L6MtDJqbWepS5NT6EW23rCuu69gwwGVFr7BpzY2MHS34mPrR0LKRqoGI19tRgpz3vFJkEueW9mQ1"; ##
SECRET-DATA
    }
    services {
        ssh {
            root-login allow;
            protocol-version v2;
        }
```

```
    }
    name-server {
        8.8.8.8;
        9.9.9.9;
    }
    arp {
        aging-timer 5;
    }
    syslog {
        file interactive-commands {
            interactive-commands any;
        }
        file messages {
            any notice;
            authorization info;
        }
    }
}
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                dhcp force-discover;
            }
        }
    }
}
protocols {
    lldp {
        interface all;
    }
    lldp-med {
        interface all;
    }
}
EOF
```

At this point, you must have created an individual Junos OS startup configuration and continuing the process.

```
# download the make-config.sh script from the Juniper CDN if you do not have it yet
# https://webdownload.juniper.net/swdl/dl/anon/site/1/record/168885.html
```

```
chmod 777 make-config.sh
./make-config.sh juniper.conf myconfig.img
.
./config/juniper.conf
Cleaning up...
removed '/var/tmp/tmp.hhQ0rcM92K/config/juniper.conf'
removed directory '/var/tmp/tmp.hhQ0rcM92K/config'
removed directory '/var/tmp/tmp.hhQ0rcM92K'
removed directory '/var/tmp/tmp.gvCkmgmvXy'
Config disk myconfig.img created
# import the junos config image to proxmox storage
qm disk import $VMID myconfig.img $VMSTORAGE --format raw | tee diskimport.txt
.
transferred 1.0 MiB of 1.0 MiB (100.00%)
transferred 1.0 MiB of 1.0 MiB (100.00%)
Successfully imported disk as 'unused0:local-lvm:vm-200-disk-1'
# extract image location from import
VMIMAGE=`cat diskimport.txt | grep "imported disk" | awk '{print $5}' | sed 's/.:/ /' | awk
'{print $2}' | sed 's/.$//'`
echo $VMIMAGE
local-lvm:vm-200-disk-1
# attach the config-image disk to the vm
qm set $VMID --ide0 $VMIMAGE,size=16M
update VM 200: -ide0 local-lvm:vm-200-disk-1,size=16M
```

Now, all our preparations are complete. You can review the resulting VM configuration.

```
# review the VM configuration made
qm config $VMID
args: -machine accel=kvm:tcg -smbios type=1,product=VM-VEX -cpu 'host,kvm=on'
boot: order=virtio0
cores: 4
ide0: local-lvm:vm-200-disk-1,size=4M
memory: 5120
meta: creation-qemu=8.1.5,ctime=1728988040
name: switch1
net0: virtio=BC:24:11:01:06:0E,bridge=vmbr0
net1: virtio=BC:24:11:6B:0B:84,bridge=ge000,firewall=0
net2: virtio=BC:24:11:7E:5C:07,bridge=ge001,firewall=0
net3: virtio=BC:24:11:FB:40:37,bridge=ge002,firewall=0
serial0: socket
smbios1: uuid=5b184467-bffe-45f3-8a4c-bb2182aa3aa5
```

```
virtio0: local-lvm:vm-200-disk-0,iothread=1,size=32524M
vmgenid: a3299ccf-293b-4df2-9458-b0fa444a9c61
```

As the VM does not contain any credentials or other limiting factors, use this VM as a template before you launch it for the first time. This allows you to launch multiple VMs as full or linked to the image clones later. Follow the steps below if you decide to proceed.

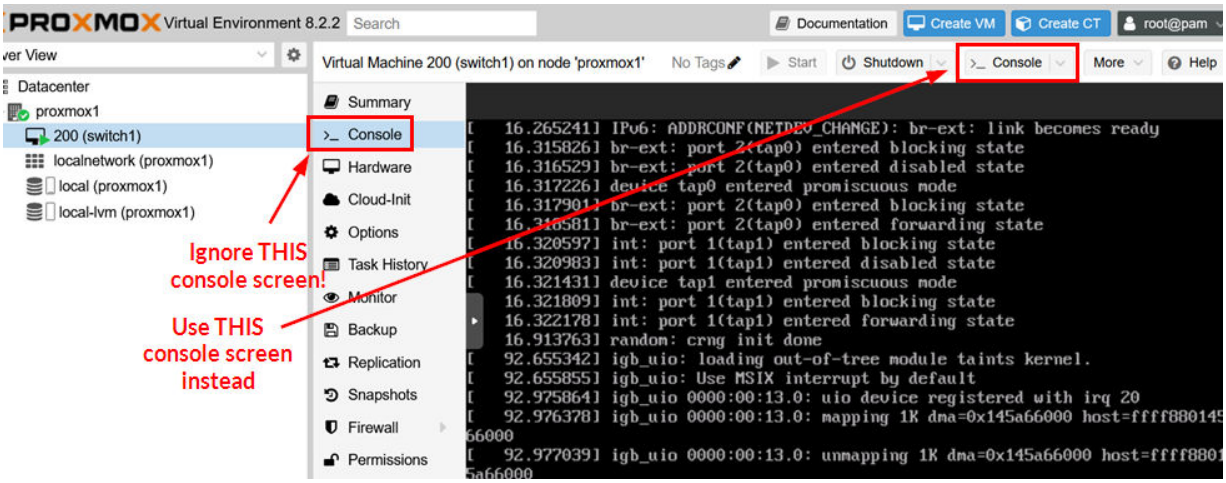
```
qm template $VMID
  Renamed "vm-200-disk-1" to "base-200-disk-1" in volume group "pve"
  Logical volume pve/base-200-disk-1 changed.
  WARNING: Combining activation change with other commands is not advised.
  Renamed "vm-200-disk-0" to "base-200-disk-0" in volume group "pve"
  Logical volume pve/base-200-disk-0 changed.
  WARNING: Combining activation change with other commands is not advised.
# select a new VMID for the clone
VMID2="201"
# create a clone of of your template VM
qm clone $VMID $VMID2 --name switch1
create linked clone of drive ide0 (local-lvm:base-200-disk-1)
  Logical volume "vm-201-disk-0" created.
create linked clone of drive virtio0 (local-lvm:base-200-disk-0)
  Logical volume "vm-201-disk-1" created.
#
# at this point you may change the interfaces assigned according to your topology
#
# review the VM configuration for the clone
qm config $VMID2
args: -machine accel=kvm:tcg -smbios type=1,product=VM-VEX -cpu 'host,kvm=on'
boot: order=virtio0
cores: 4
ide0: local-lvm:vm-201-disk-0,size=4M
memory: 5120
meta: creation-qemu=8.1.5,ctime=1729094281
name: switch1
net0: virtio=BC:24:11:87:61:1B,bridge=vmbr0
net1: virtio=BC:24:11:B2:11:52,bridge=ge000,firewall=0
net2: virtio=BC:24:11:79:0C:A1,bridge=ge001,firewall=0
net3: virtio=BC:24:11:DF:BC:BF,bridge=ge002,firewall=0
serial0: socket
smbios1: uuid=b81068a9-8f7e-423a-bbb8-7738da5f98df
virtio0: local-lvm:vm-201-disk-1,iothread=1,size=32524M
```

```
vmgenid: de47f143-5f48-44c4-8674-beb7d1b91bd2
# start the clone vJunos-switch VM
qm start $VMID2
brctl show
bridge name     bridge id               STP enabled     interfaces
ge000           8000.de5f3a3d3a9b       no              tap201i1
ge001           8000.da08ff719f7c       no              tap201i2
ge002           8000.8614a67130b7       no              tap201i3
vmbr0           8000.5847ca7543fe       no              enp2s0
                                                        tap201i0
```

If you have decided not to use a template/clone yet, then start the first vJunos-switch VM for testing now.

```
# start the new vJunos-switch VM
qm start $VMID
# review the linux-bridges and attached interfacs for our first VM
brctl show
bridge name     bridge id               STP enabled     interfaces
ge000           8000.0ac0df72ec4b       no              tap200i1
ge001           8000.623437ae4bac       no              tap200i2
ge002           8000.72c0fc5f9933       no              tap200i3
vmbr0           8000.5847ca7543fe       no              enp2s0
                                                        tap200i0
```

You can now review the VM console in the Proxmox GUI. Ensure you use the correct button to avoid any changes to the outer VM screen on the Routing Engine. The Routing Engine is where all the Junos OS configuration starts and has its own screen. See the figure below for the Console options to select.



Linux Bridge and VM Interface Post VM Creation Changes on Proxmox VE

Launching the vJunos-switch VM does not meet the needs of most labs. You must tweak the standard Linux bridge used in the example after every new VM launch. For the detailed explanation, see chapter "Linux Bridge and VM Interface Post VM Creation Changes" on page 37. Hence, you do not need to repeat it here. EVE-NG automatically manages these tweaks.

Proxmox VE does not provide VM interfaces details and their names through CLI locally. However, these details are available in the REST API for the GUI. Using the provided command `pvesh`, you can easily access the VM interface and extract JSON based information about the created VM interfaces. Hence, it is easier to rebuild a new script `vm-bridge-update.sh` using `pvesh` and `jq` commands and regular BASH programming. See the instructions as shown below.

```
apt-get install jq
rm -f vm-bridge-update.sh
touch vm-bridge-update.sh
chmod 777 vm-bridge-update.sh
vi vm-bridge-update.sh
```

Copy and paste the below configuration to your editor. Then, save and close.

```
#!/bin/bash
# use API to get first nodename
pvesh get /nodes --output-format json | jq -r '.[].node' >nodes.txt
VMNODE=`cat nodes.txt | head -1`
echo 'We run this on node: '$VMNODE
# use API to get nic interfaces of our VM
pvesh get /nodes/$VMNODE/qemu/$1/status/current --output-format json | jq -r '.nics | keys[]'
>/tmp/vminterfacelist.txt
# ignore first interface fxp0
cat /tmp/vminterfacelist.txt | tail -n +2 >/tmp/vminterfacelist2.txt
#cat /tmp/vminterfacelist2.txt
while IFS= read -r line
do
  INTERFACE="$line"
  #echo $INTERFACE
  BRIDGE=`find /sys/devices/virtual/net -name $INTERFACE | grep '/brif/' | sed 's/\// /g' | awk
'{print $5}'`
  # change MTU to higher value
```

```
  RUNME="ip link set dev "$INTERFACE" mtu 9200"
  echo $RUNME
  eval $RUNME
  # enable LLDP and 802.1x on bridge
  RUNME="echo 65528 > /sys/class/net/"$BRIDGE"/bridge/group_fwd_mask"
  echo $RUNME
  eval $RUNME
  # enable LACP on link
  RUNME="echo 16388 > /sys/class/net/"$INTERFACE"/brport/group_fwd_mask"
  echo $RUNME
  eval $RUNME
done < /tmp/vminterfacelist2.txt
num=0
while IFS= read -r line
do
  INTERFACE="$line"
  BRIDGE=`find /sys/devices/virtual/net -name $INTERFACE | grep '/brif/' | sed 's/\// /g' | awk
'{print $5}'`
  MTU=`cat /sys/class/net/$BRIDGE/mtu`
  if [ "$MTU" != "9200" ]; then
    echo 'Warning! Bridge:'$BRIDGE' did not follow new MTU setting of interface:'$INTERFACE'
check other interfaces attached to same bridge and correct please!'
    num=1
  fi
done < /tmp/vminterfacelist2.txt
exit $num
```

With the new script, you can now update the Linux bridges and interfaces of the VM after it is started. The selected API's first node is suitable for a single Proxmox VE installation. If you have a cluster, you might need to change the above script.

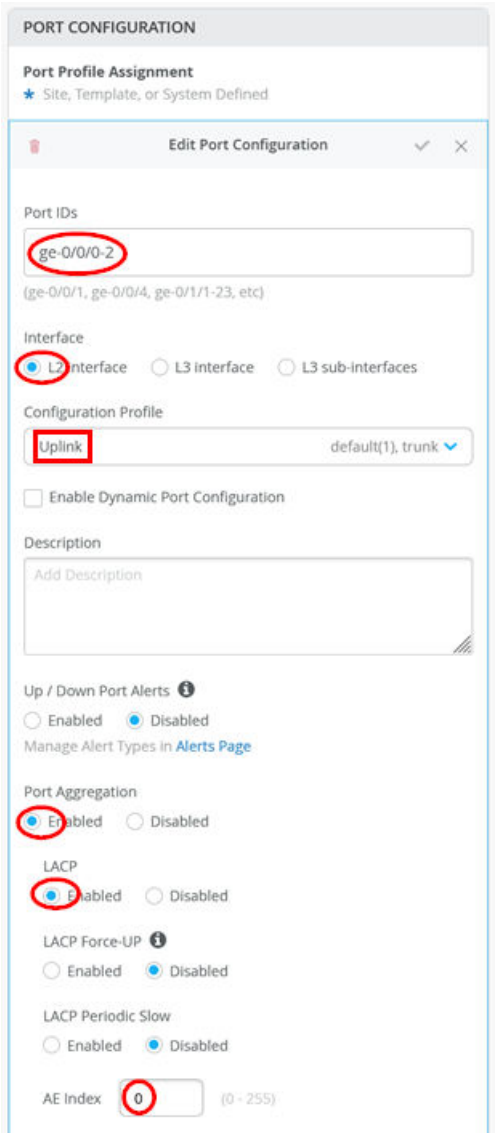
```
./vm-bridge-update.sh $VMID
We run this on node: proxmox1
ip link set dev tap200i1 mtu 9200
echo 65528 > /sys/class/net/ge000/bridge/group_fwd_mask
echo 16388 > /sys/class/net/tap200i1/brport/group_fwd_mask
ip link set dev tap200i2 mtu 9200
echo 65528 > /sys/class/net/ge001/bridge/group_fwd_mask
echo 16388 > /sys/class/net/tap200i2/brport/group_fwd_mask
ip link set dev tap200i3 mtu 9200
```

```
echo 65528 > /sys/class/net/ge002/bridge/group_fwd_mask
echo 16388 > /sys/class/net/tap200i3/brport/group_fwd_mask
```

To validate the first test for your Linux bridge enhancements, check for LLDP neighbor announcements from your vJunos-switch VM. With the `juniper.conf` instructions but without the tweak, you do not see the announcements using `tcpdump`). See the example below.

```
root@proxmox1:~# tcpdump -eni ge000
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ge000, link-type EN10MB (Ethernet), snapshot length 262144 bytes
13:47:37.917669 bc:24:11:6b:0b:84 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc), length 322:
LLDP, length 308: vjunos
13:48:07.692425 bc:24:11:6b:0b:84 > 01:80:c2:00:00:0e, ethertype LLDP (0x88cc), length 322:
LLDP, length 308: vjunos
```

To perform a final test, launch a second vJunos-switch connected 1:1 to the first VM. Then, establish a LAG with active LACP between the two VMs. The configuration for both virtual switches in the Mist Cloud GUI is shown below.



If you inspect locally on the vJunos-switch console, you should see LLDP neighbors and the established LACP links between the two switches. This step verifies that your lab works as expected.

```
root@switch1> show lldp neighbors
Local Interface    Parent Interface    Chassis Id          Port info          System Name
ge-0/0/0           ae0                 2c:6b:f5:3b:3b:c0   ge-0/0/0
switch2
ge-0/0/1           ae0                 2c:6b:f5:3b:3b:c0   ge-0/0/1
switch2
ge-0/0/2           ae0                 2c:6b:f5:3b:3b:c0   ge-0/0/2
switch2
root@switch1> show lacp interfaces
Aggregated interface: ae0
```

```
    LACP state:       Role   Exp   Def  Dist  Col  Syn  Aggr  Timeout  Activity
      ge-0/0/0       Actor    No    No   Yes  Yes  Yes   Yes     Fast    Active
      ge-0/0/0     Partner    No    No   Yes  Yes  Yes   Yes     Fast    Active
      ge-0/0/1       Actor    No    No   Yes  Yes  Yes   Yes     Fast    Active
      ge-0/0/1     Partner    No    No   Yes  Yes  Yes   Yes     Fast    Active
      ge-0/0/2       Actor    No    No   Yes  Yes  Yes   Yes     Fast    Active
      ge-0/0/2     Partner    No    No   Yes  Yes  Yes   Yes     Fast    Active
    LACP protocol:        Receive State  Transmit State          Mux State
      ge-0/0/0                  Current   Fast periodic Collecting distributing
      ge-0/0/1                  Current   Fast periodic Collecting distributing
      ge-0/0/2                  Current   Fast periodic Collecting distributing
```

AMD-CPU Unofficial Tweaks

NOTE: You can use the following tweaks to run vJunos-switch VM on x86 based AMD CPUs. Juniper engineering has not tested vJunos-switch with non-Intel CPUs. Try these in official tweaks with no guaranteed success.

AMD CPU EVE-NG (raw Qemu) tweaks for vJunos-switch VM in *.yaml-file `qemu_options`:

```
-smbios type=1,product=VM-VEX
-cpu IvyBridge,ibpb=on,spec-ctrl=on,ssbd=on,virt-ssbd=on,svm=on,erms=off
```

AMD CPU KVM virt-install CLI tweaks for vJunos-switch VM.

```
--sysinfo smbios,system.product=VM-VEX
--cpu IvyBridge,require=svm,disable=erms
```

AMD CPU KVM virt-manager GUI XML tweaks for vJunos-switch VM.

```
  <sysinfo type='smbios'>
    <system>
      <entry name='product'>VM-VEX</entry>
    </system>
  </sysinfo>
```

```
  <os>
    <type arch='x86_64' machine='pc-i440fx-focal'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <cpu mode='custom' match='exact' check='full'>
    <model fallback='forbid'>IvyBridge</model>
    <feature policy='require' name='ibpb'/>
    <feature policy='require' name='spec-ctrl'/>
    <feature policy='require' name='ssbd'/>
    <feature policy='require' name='virt-ssbd'/>
    <feature policy='require' name='svm'/>
    <feature policy='disable' name='erms'/>
    <feature policy='require' name='hypervisor'/>
    <feature policy='require' name='arat'/>
    <feature policy='require' name='xsaveopt'/>
  </cpu>
```

Access Point Integration

If your lab needs to have both Wi-Fi Access Point (AP) and WLAN clients, then we recommend the following configuration as a best practice. The AP must be a physical hardware to emit radio waves for the WLAN client tests. We recommend that you allocate a free Ethernet port at the server for each AP. For each physical Ethernet interface at the server, create a unique Linux bridge assigning the NIC. Then, assign the individual Linux bridge to a vJunos-switch interface for integration. As your server NIC usually has no PoE option, you need a power supply for the AP.

You can now test any available physical WLAN clients.

You can virtualize the wireless client on your server, which is useful for remote lab access. A VM can be built as a Linux or Windows Wireless client. For the virtual client's Wi-Fi Radio, we recommend using an inexpensive WLAN USB adapter connected to the server with the client VM. Identify the USB ID of the adapter and map it to the VM when starting. The following example shows how to archive this in an Ubuntu KVM libvirtd environment.

```
# If you have multiple WLAN Clients then you may see more than one adapter to use
# with just one adapter the number is "1"
ADAPTERNO="1"
# we use the lsusb command to retrieve the WLAN adapter
```

```
# however sometime the vendor give the adapter a different name
# in this case you may need editing the key-words we search for
ADAPTERSEARCH="WLAN|Wireless|802.11"
lsusb | egrep -i $ADAPTERSEARCH >usbwlan.txt
line=`sed -n $ADAPTERNO'p' usbwlan.txt`
USBBUS=`echo $line | awk '{print $2}'`
USBDEVICE=`echo $line | awk '{print $4}' | sed 's/://g'`
echo 'WLAN Adapter at Bus:'$USBBUS' Device:'$USBDEVICE
# do not proceed if no WLAN adapter was detected
# start the client VM with adding the WLAN adapter as hostdev-device
virt-install -n desktopvm --vcpus=1 -r 2048 \
--hvm --os-variant ubuntu20.04 --cpu host \
--import \
--disk path=/var/lib/libvirt/images/desktopvm.qcow2,format=qcow2 \
--controller usb2 --hostdev $USBBUS.$USBDEVICE \
--graphics vnc,listen=0.0.0.0 -noautoconsole
```

NOTE: If you reboot the server after the client VM is started, the WLAN adapter USB IDs might change.