A VMX INTALLATION GUIDE

http://www.junosandme.net

By David Roy

Installing VMX for lab simulation

Let's start the first VMX installation by the simplest use case. VMware ESXi offers a graphical interface through the vSphere Client to create and manage your Virtual Machine and your virtual bridges. This is the simplest way to quickly create several VMX routers for lab testing purposes. The aim is to deploy this topology based on two VMX routers on our ESXi server:

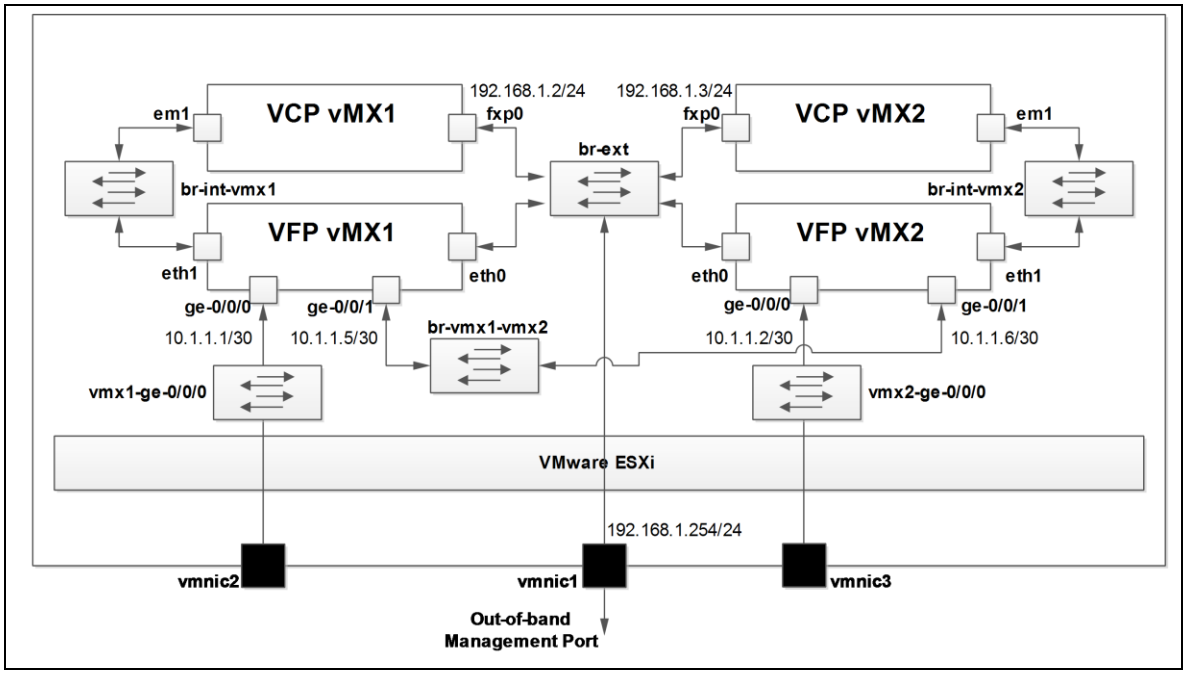


Figure . VMX topology on ESXi

Server and host OS requirements

There are only few hardware and software requirements for low-bandwidth application which are:

- Processor has to support VT-X. All recent x86 (Intel or AMD) processors support today standard Virtualization Technique).
- Make sure your server has enough memory and cpu capacities to install at least one VMX instance. For that refer to the table X-X.
- The VMware ESXi version must be at least 5.5.0u2

ESXi installation

The installation of the VMware ESXi is out of the scope of this book. Just simply create a bootable USB key based on the ISO image of the ESXi. Then, follow the step by step installation procedure. No specific option is required to run VMX on ESXi.

Organize your "work folder" on ESXi

With the vSphere client you have access to your ESXi. You can organize your datastore as followed (this is just a recommendation):

1. Access to the datastore: from the summary tab right click on the datastore and select Browse Datastore:

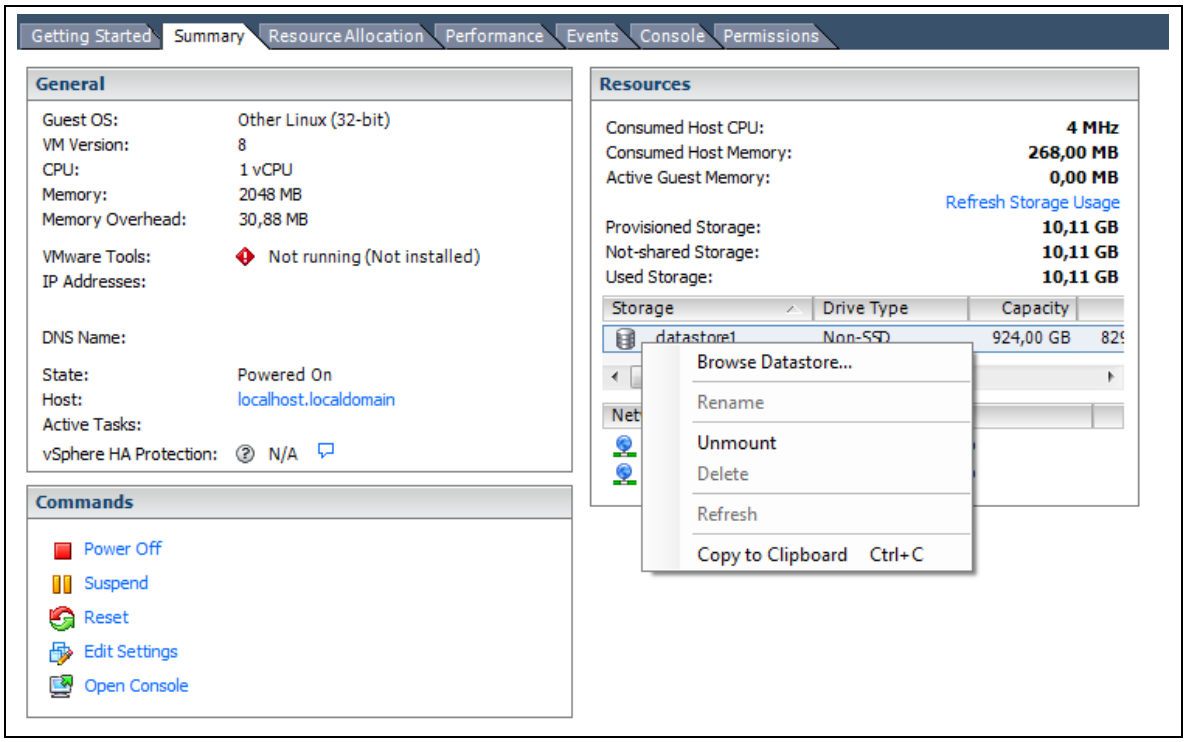


Figure . Access to the ESXi datastore

2. Then create one folder per VMX router – here we have created vmx1 and vmx2 folders. The others folder referring to vmx1 and vmx2 will create automatically during the VM deployment.

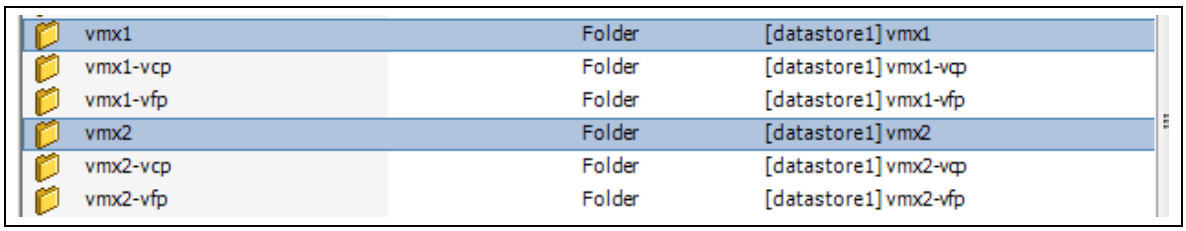


Figure . How to organize your work folder

You can also create a Resource Pool without any restriction in order to merge all your Virtual Machines into a single container. For that, right click at the root level of the server and select New Resource Pool.

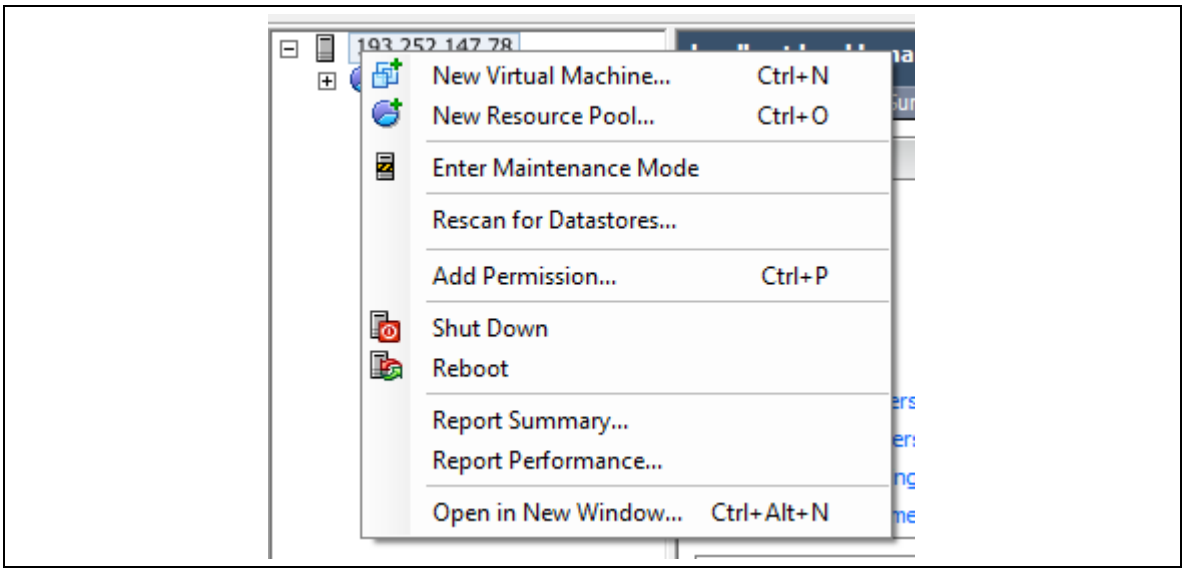


Figure . Create a new resource pool

On the next window, just fill a name – here vRouter:

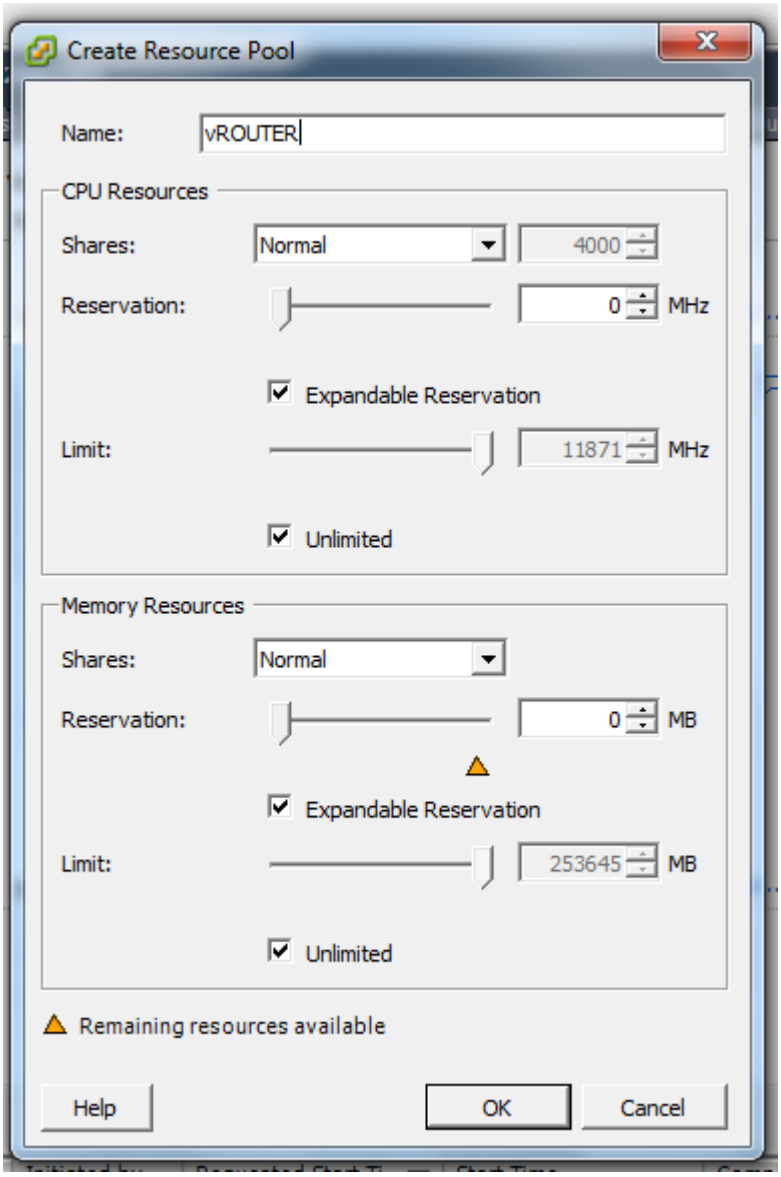


Figure . Adding a resource pool for your virtual lab

Preparing the virtual bridges

As shown on the Figure , there are several virtual bridges needed in our topology:

- br-ext: to interconnect out-of-band management interfaces of the VCP and VFP of both VMX and the physical port vmnic1 of the server.
- br-int-vmx1: to connect the VCP and VFP virtual machines of vmx1
- br-int-vmx2: to connect the VCP and VFP virtual machines of vmx2
- br-vmx1-vmx2: to connect the interface ge-0/0/1 of the two VMX routers.
- vmx1-ge-0/0/0: to connect the interface ge-0/0/0 of the vmx1 to the physical interface vmnic2
- vmx2-ge-0/0/0: to connect the interface ge-0/0/0 of the vmx2 to the physical interface vmnic3

We highly recommend to create all the required virtual bridges before starting the installation of the VMX instances. To create the br-ext virtual bridge just follows these steps:

1. Click on the "Configuration" tab, then select "Networking" and "Add Networking":

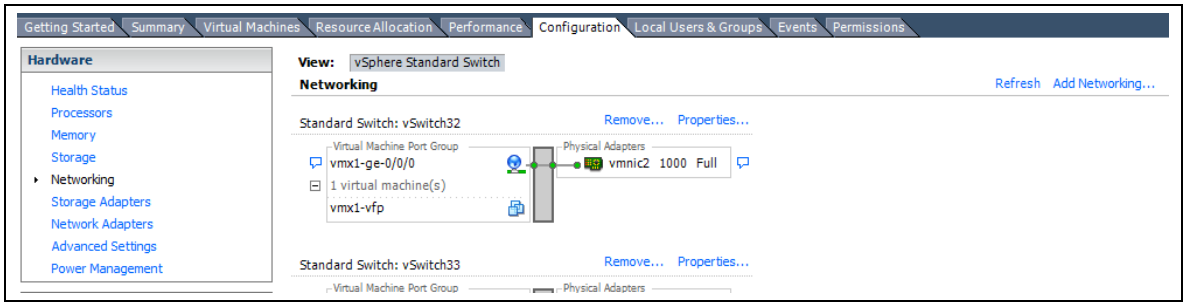


Figure . Create a virtual bridge with vSphere Client

2. Choose the Connection Types as Virtual Machine

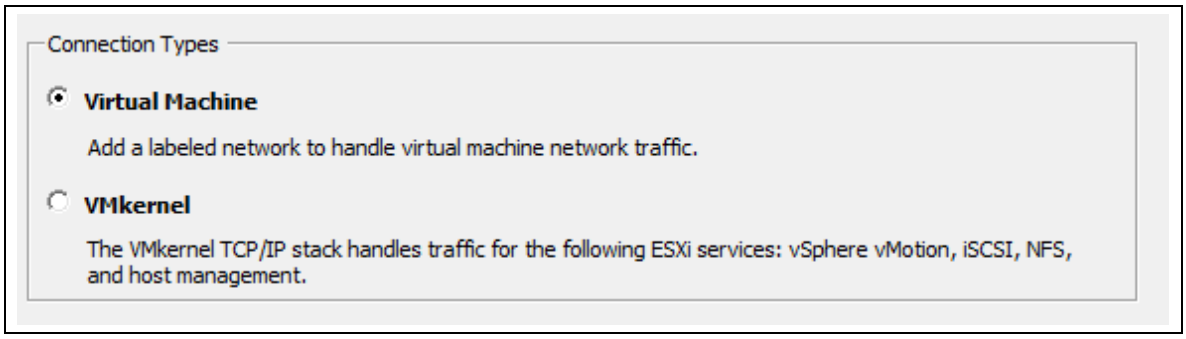


Figure . Select the connection type

3. Create a vSphere standard switch with vmnic1 selected:

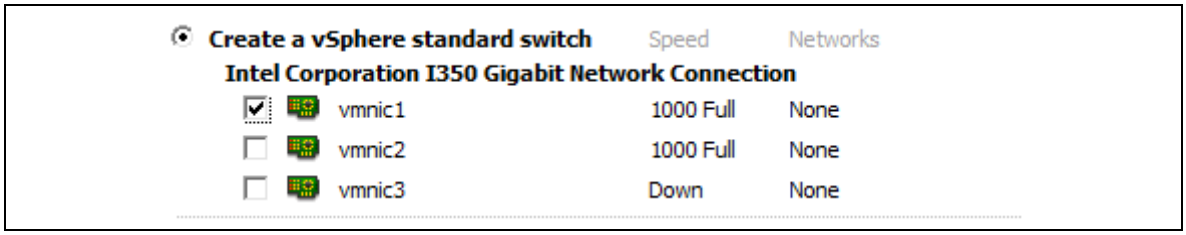


Figure . Add a physical NIC to the bridge

4. Add a name to your virtual bridge: br-ext and then finish.

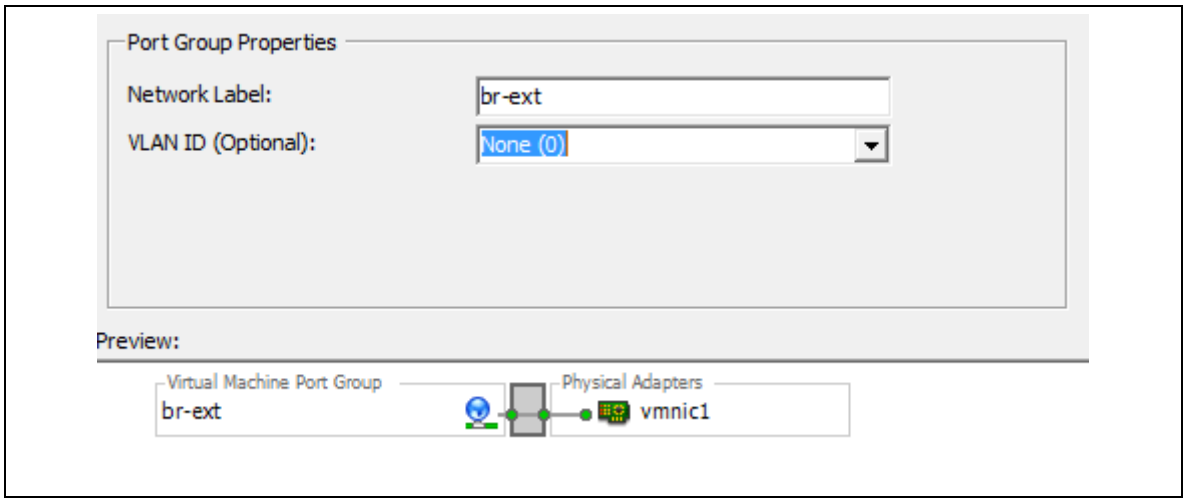


Figure . Add a name for your bridge

5. Then scroll down to find the br-ext bridge and click to Properties.

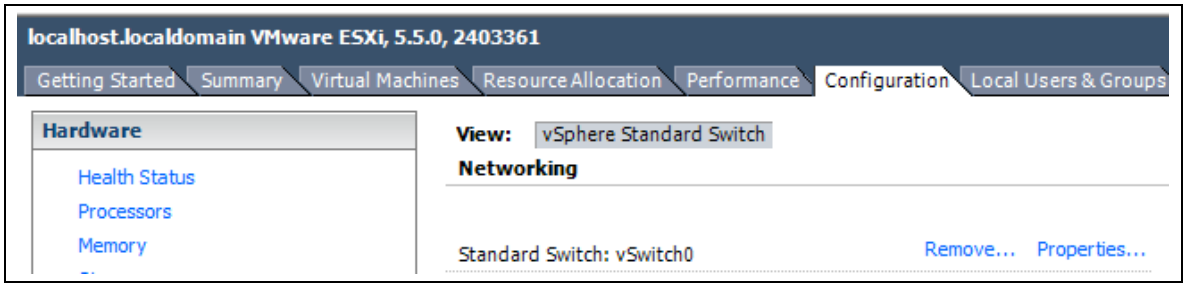


Figure . Modify the properties of an existing bridge

6. Select vSwitch and then click Edit on bottom.

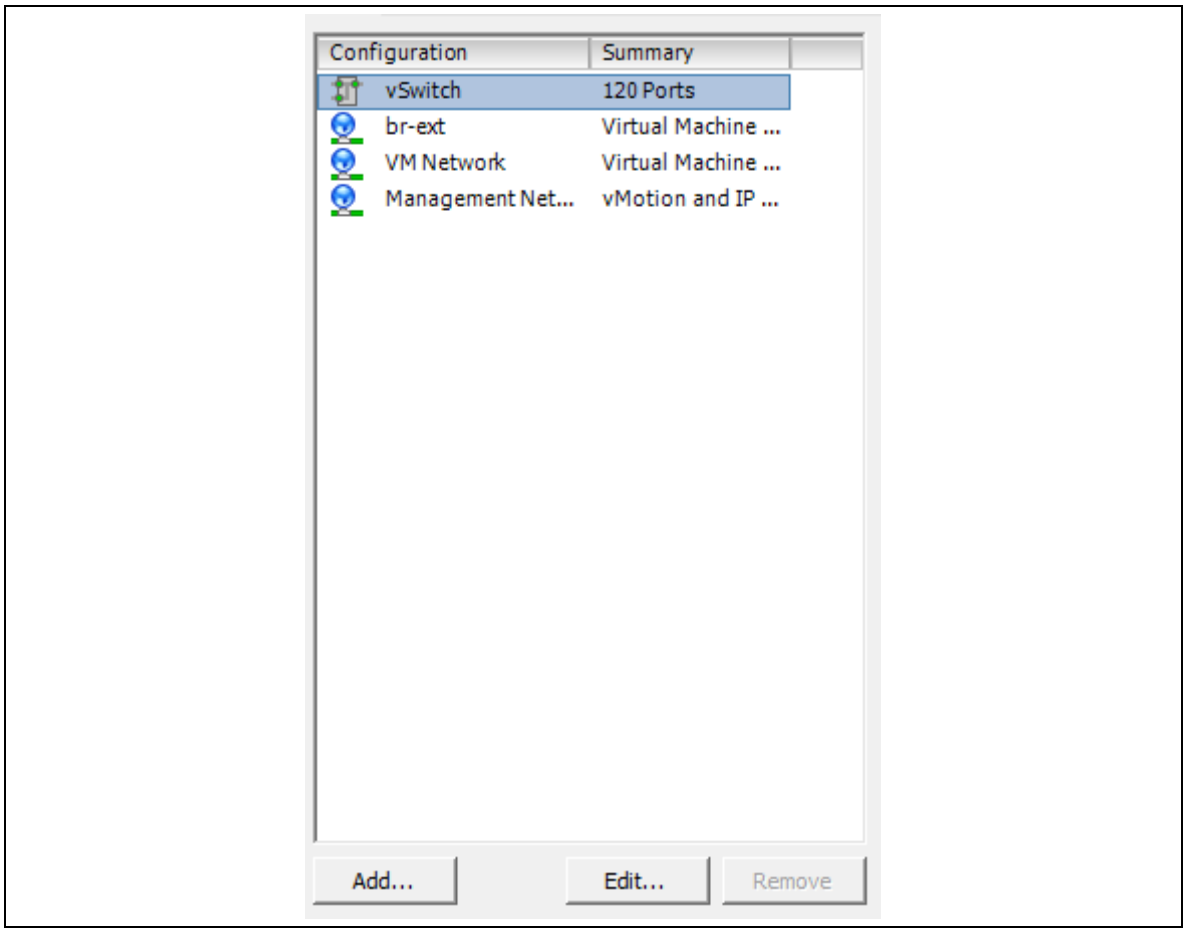


Figure . Select and edit properties of a vSwitch

7. Configure the MTU to 9000 in the General Tab and Promiscuous Mode to Accept on Security Tab

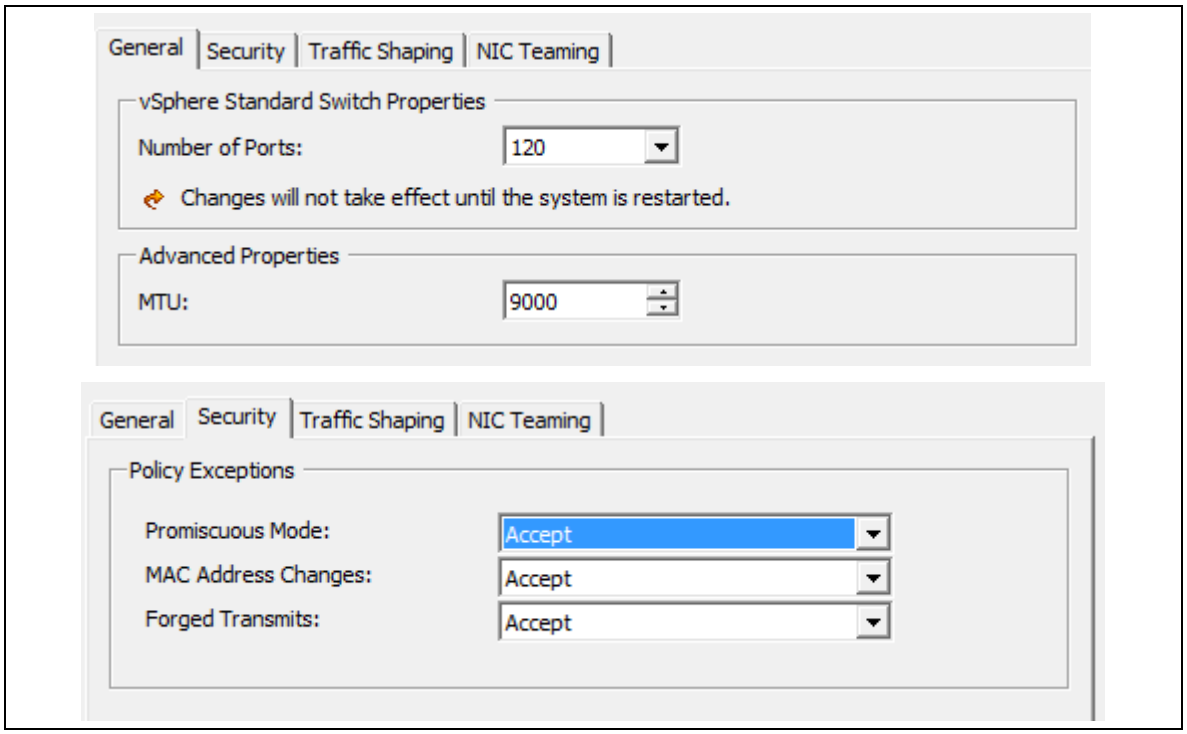


Figure . Change MTU and Security options of an existing bridge

You have just to repeat these steps to create the other virtual bridges. Just notice, at the step 3, depending on the virtual bridge you might have to attach a vmnic (vmnic2 or 3 for vmx1-ge-0/0/0 and vmx2-ge-0/0/0) or unselect all vmnic when the virtual bridge attaches only purely virtual interface (this is the case for br-int-vmx1, br-int-vmx2 and br-vmx1-vmx2). Don't forget to modify the MTU and Promiscuous Mode for each virtual bridge. Finally you should have the following bridges created:

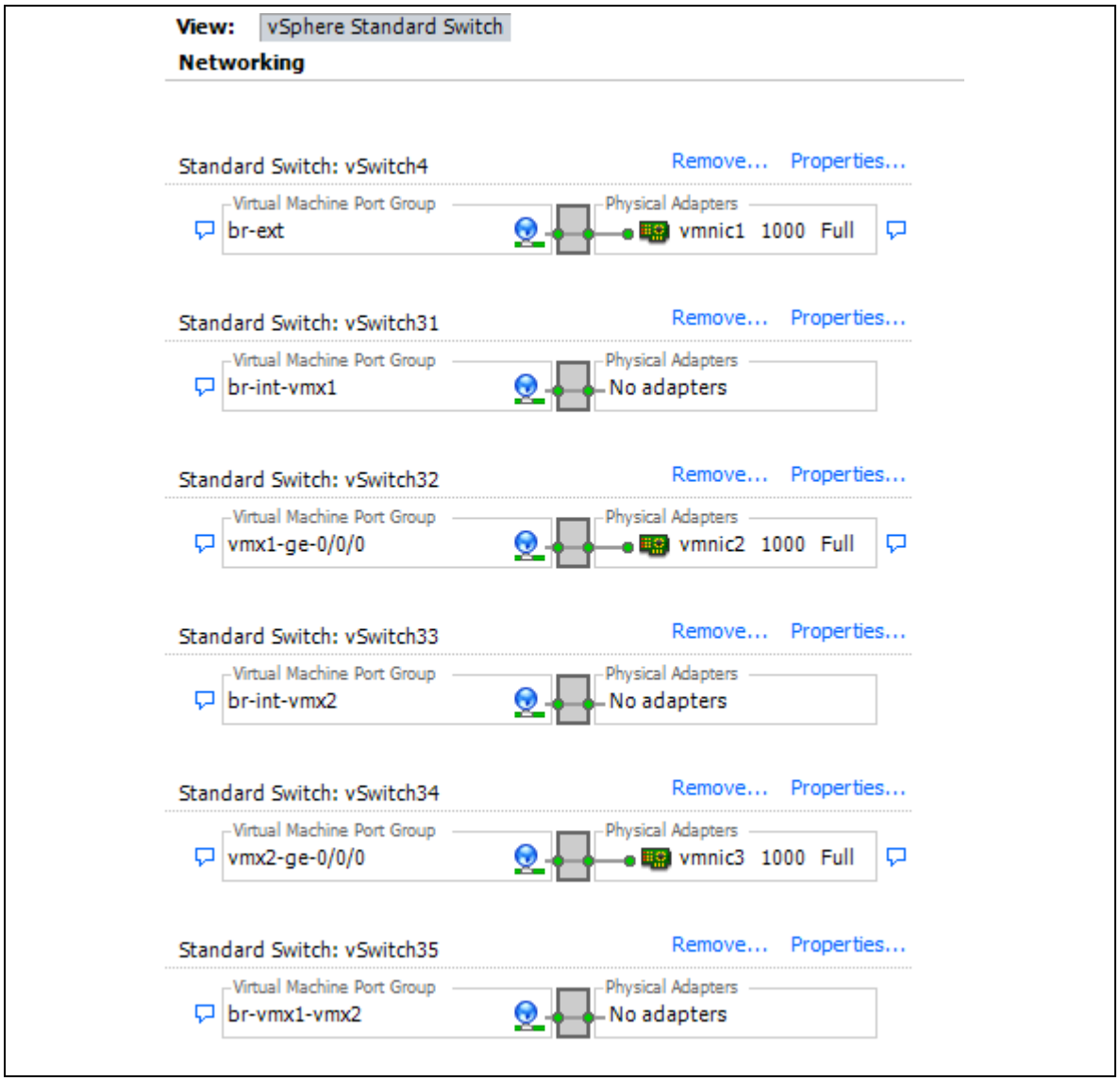


Figure . General view of all vSwitch

Installing VCP VM

First of all, retrieve the VMX package for ESXi on the Juniper website. For us it is installation package is vmx-esxi-15.1F4.15. Decompress the package on your computer and upload the four files of the vmx-15.1F4-3-ESXi\vmdk into the vmx1 folder of the datastore:

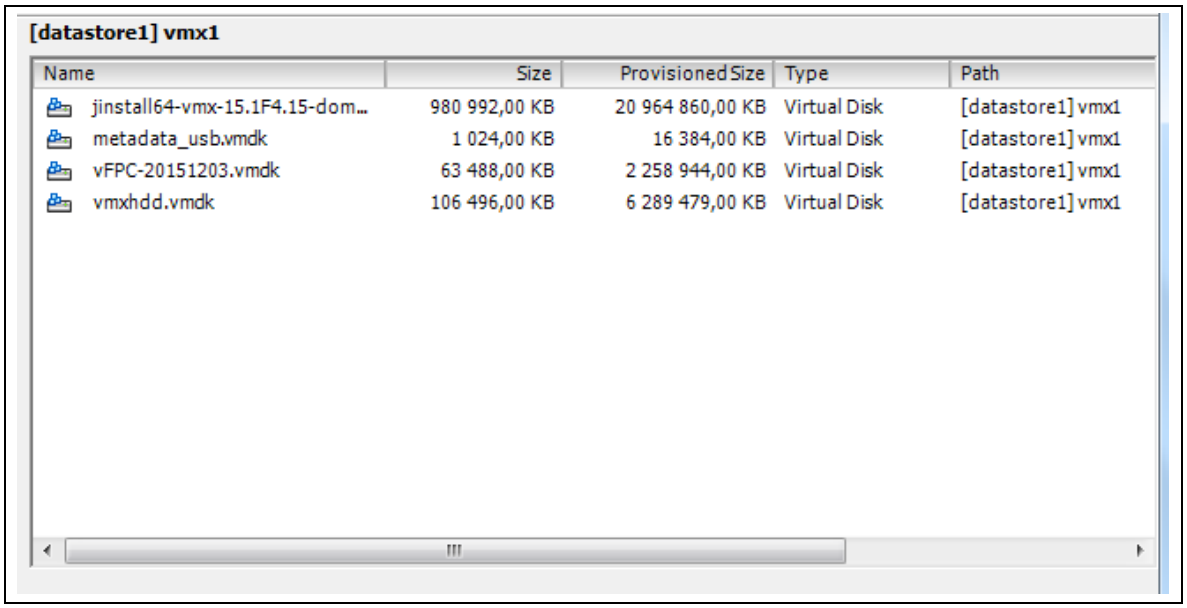


Figure . Upload the VMX files in the datastore

Follow this step by step procedure to create the VCP virtual machine.

1. Click on Create a new Virtual Machine
2. Select "Custom"
3. Choose a name for your Virtual Machine: here vmx1-vcp

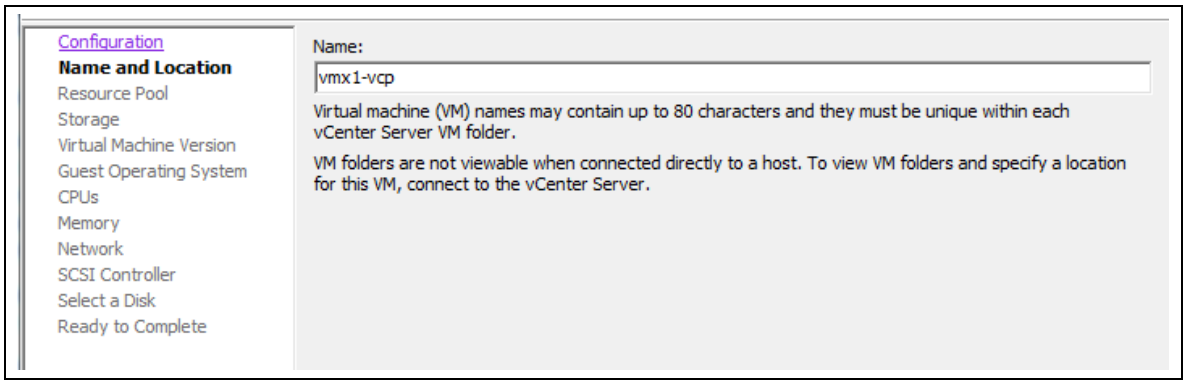
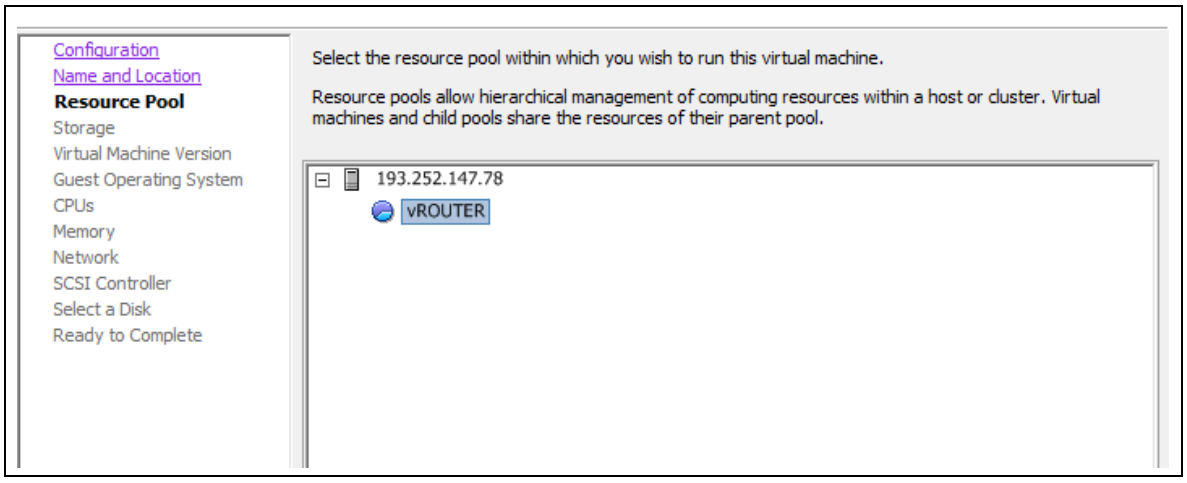


Figure . Create the VCP VM

4. Select the target Resource Pool: here vRouter



** Figure . Assign the VM to a resource pool*

5. Change nothing regarding the Storage information, just click next
6. Select at least a version 8 for the virtual Machine
7. Choose the type of Guest OS as Other 64bits

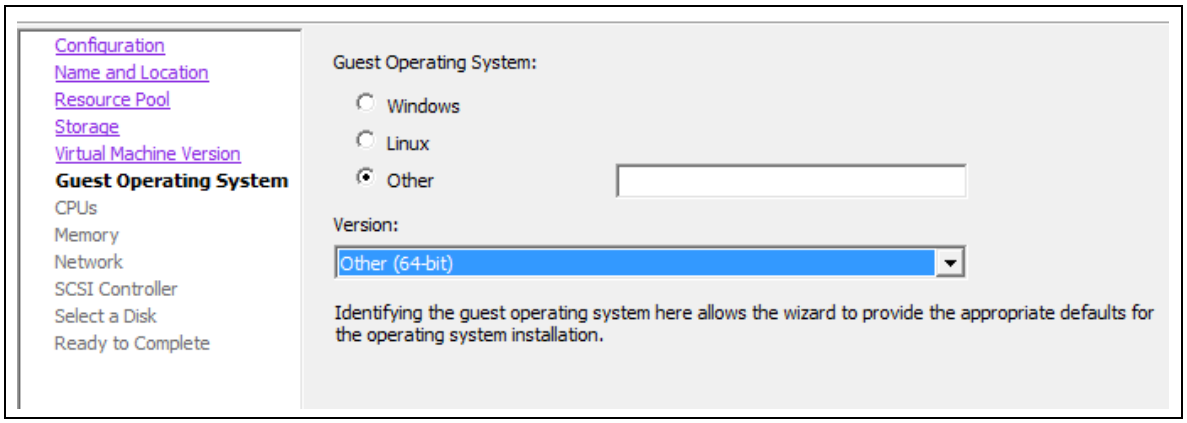


Figure . Select the Guess OS type

8. Assign 1 socket and 1 vCPU to the VCP

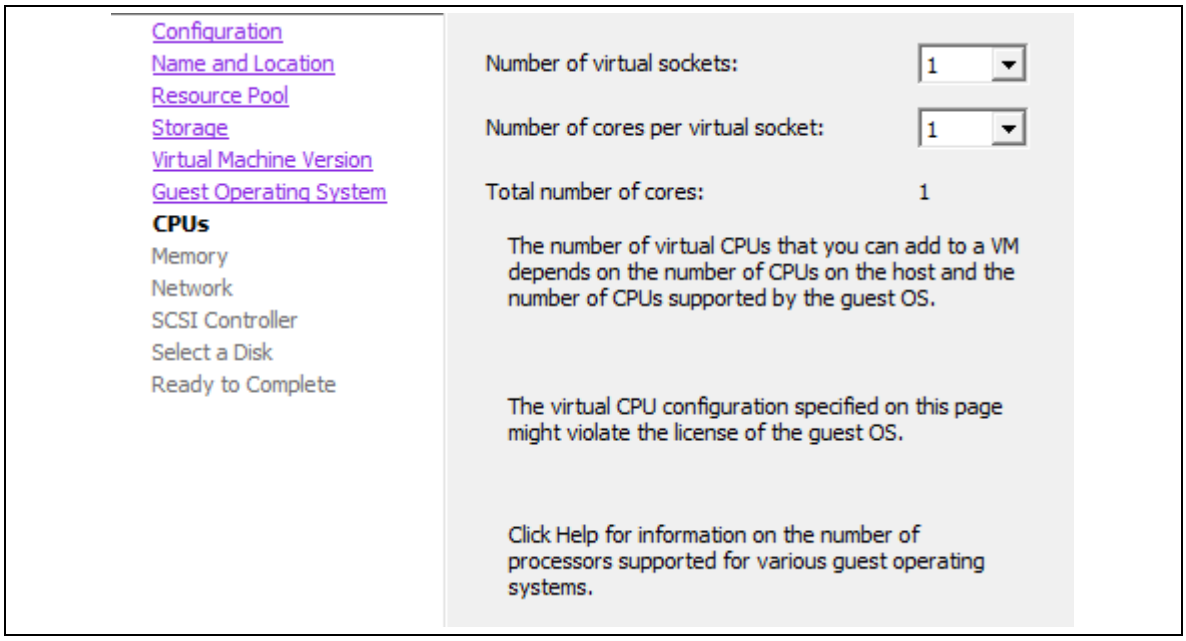


Figure . Assign vCPU to VCP VM

9. Assign 2GB of memory:

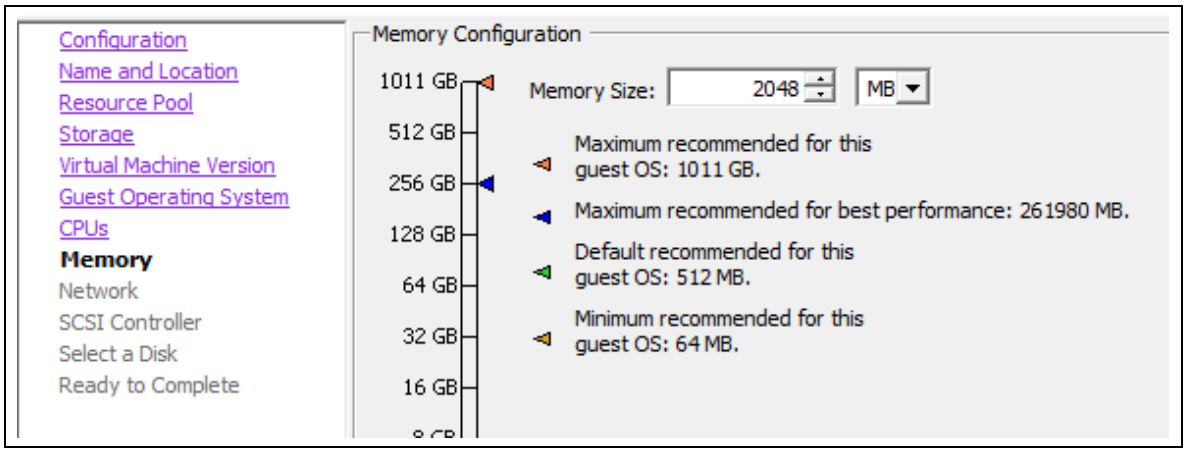


Figure . Assign memory to VCP VM

10. For interface, select 2 virtual interfaces – the first one will be attached to the fxp0 of the VCP and the second one to the em1 interface. Therefore, connect the first virtual interface to the bridge br-ext and the second interface to the bridge br-int-vmx1. For both interface choose the E1000 type:

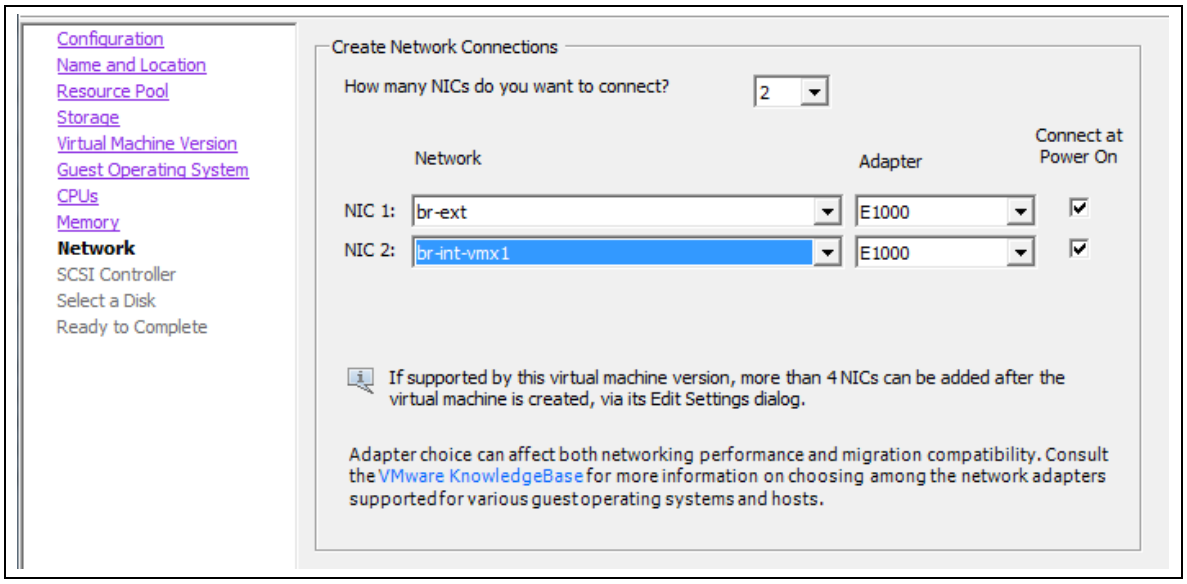


Figure . Create virtual interfaces of VCP

11. Let the LSI Logic Parallel selected as SCSI controller
12. Regarding the Hard Disk choose Use Existing virtual Disk:

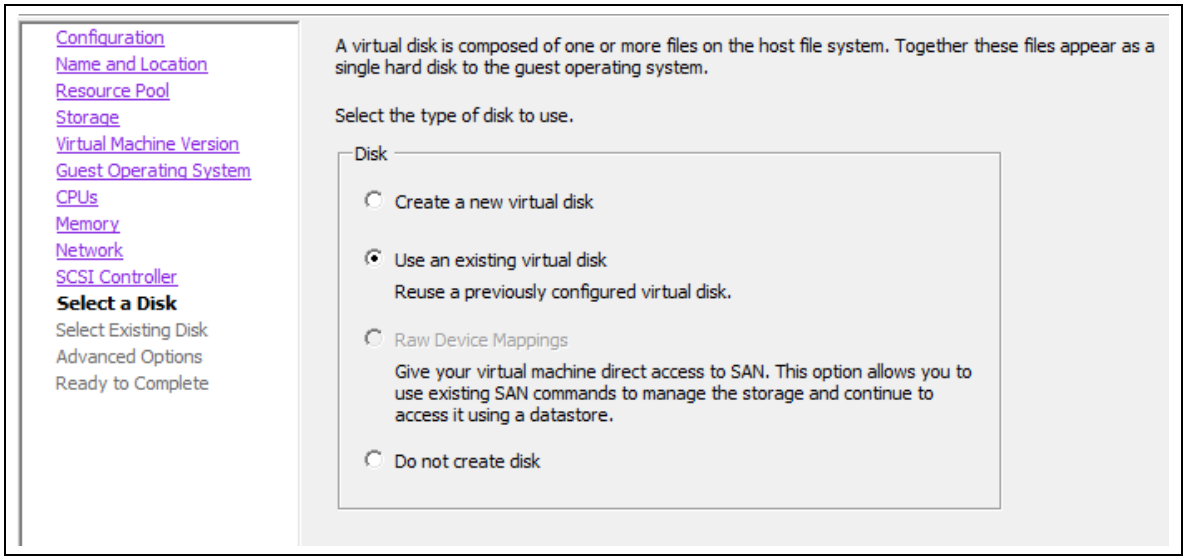


Figure . Configure the master virtual disk of VCP

13. Then choose the following disk into the vmx1 folder: jinstall64-vmx-15.1F4.15-domestic.vmdk

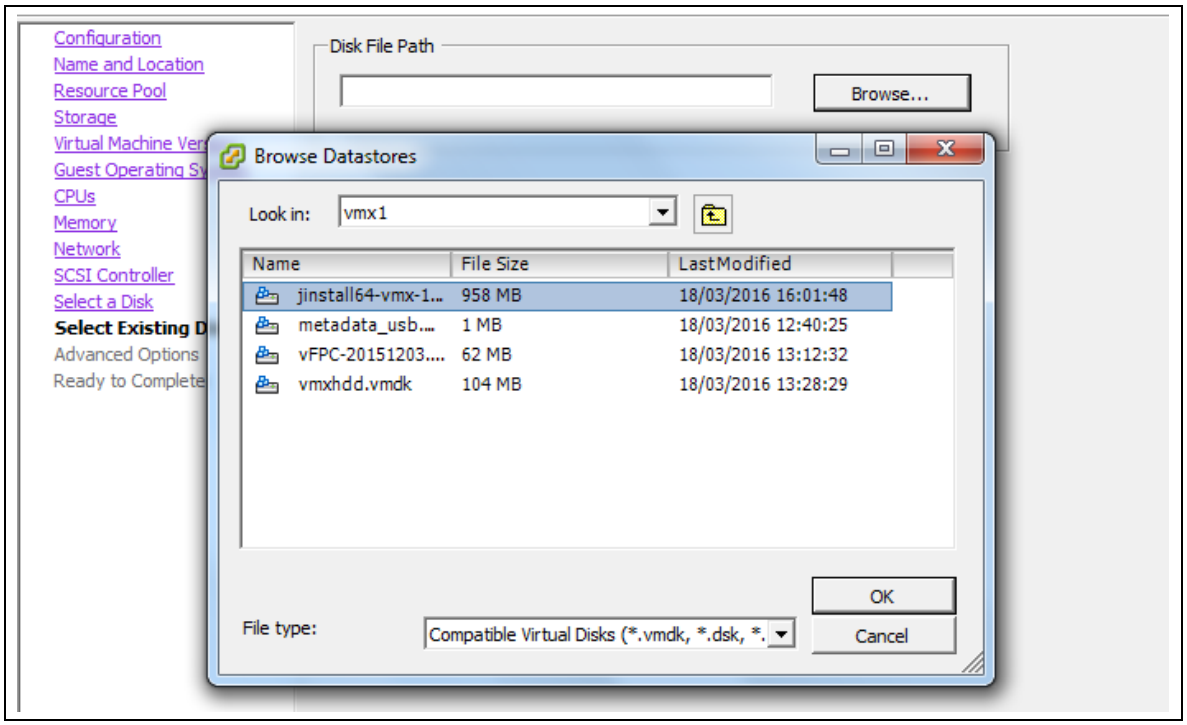


Figure . Select image from datastore

14. On the Advance Option tab click "Edit Before" at the bottom of the screen then click Continue

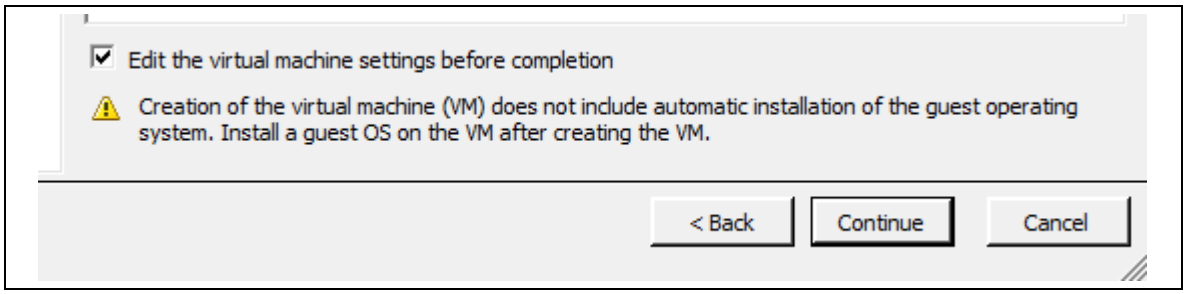


Figure . Advance editing of the VCP VM

15. Add a second Hard Disk, choose one more time Use Existing virtual Disk and select the file vmxhdd.vmdk into the vmx1 folder of the datastore:

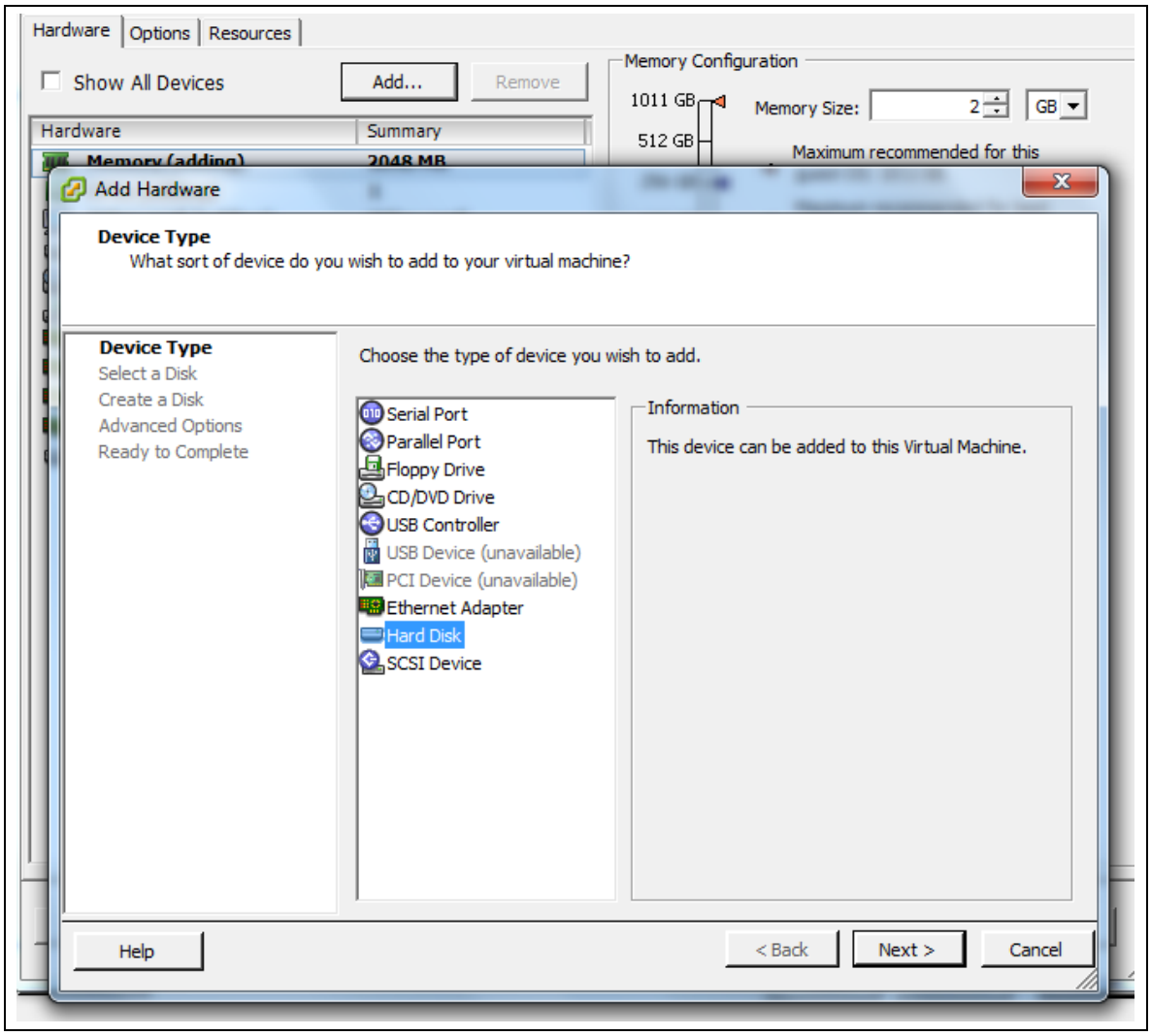


Figure . Add a second hard disk to VCP

16. Repeat the step one more time: Add a third Hard Disk, choose once again Use Existing virtual Disk and select the file metadata_usb.vmdk into the vmx1 folder of the datastore
17. Then finish the installation of the virtual machine

Installing VFP VM

Follow this step by step procedure to create the VFP virtual machine.

1. Click on Create a new Virtual Machine
2. Select “Custom”
3. Choose a name for your Virtual Machine: here vmx1-vfp

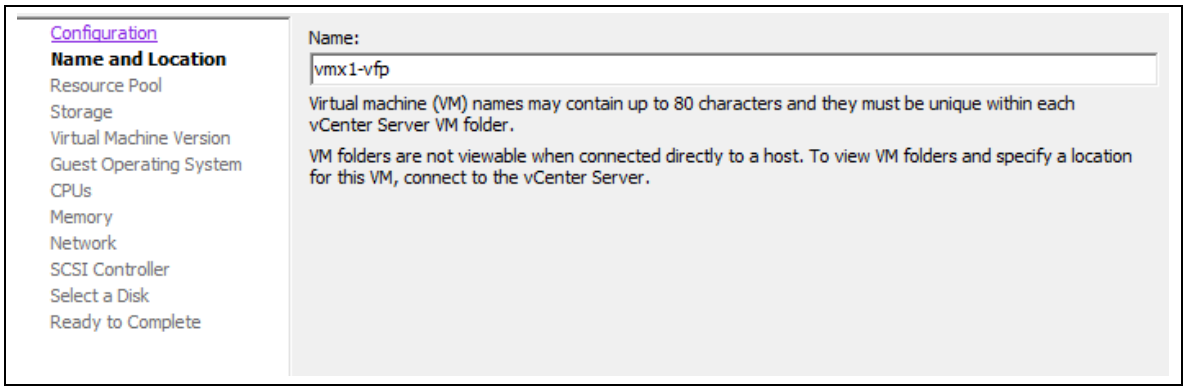


Figure . Create the VFP VM

4. Select the target Resource Pool: here vRouter
5. Change nothing regarding the Storage information, just click next
6. Select at least a version 8 for the virtual Machine
7. Choose the type of Guest OS as Other 64bits
8. Assign 1 socket and 3 vCPU to the VFP

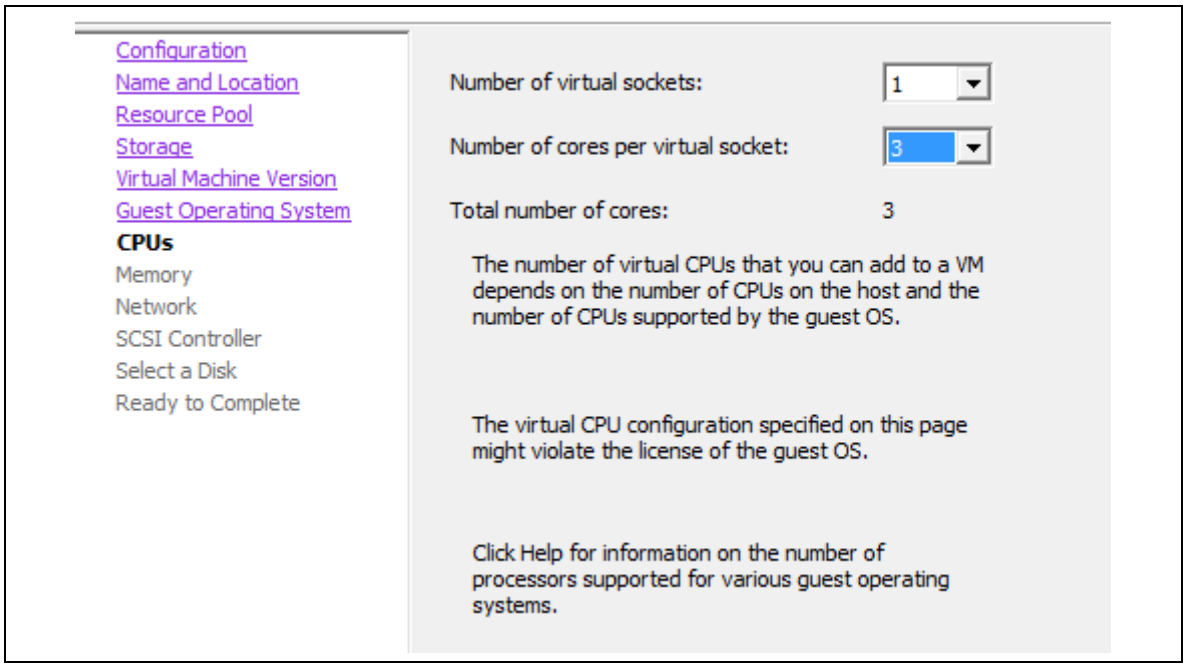


Figure . Assign vCPU to VFP VM

9. Assign 8GB of memory:

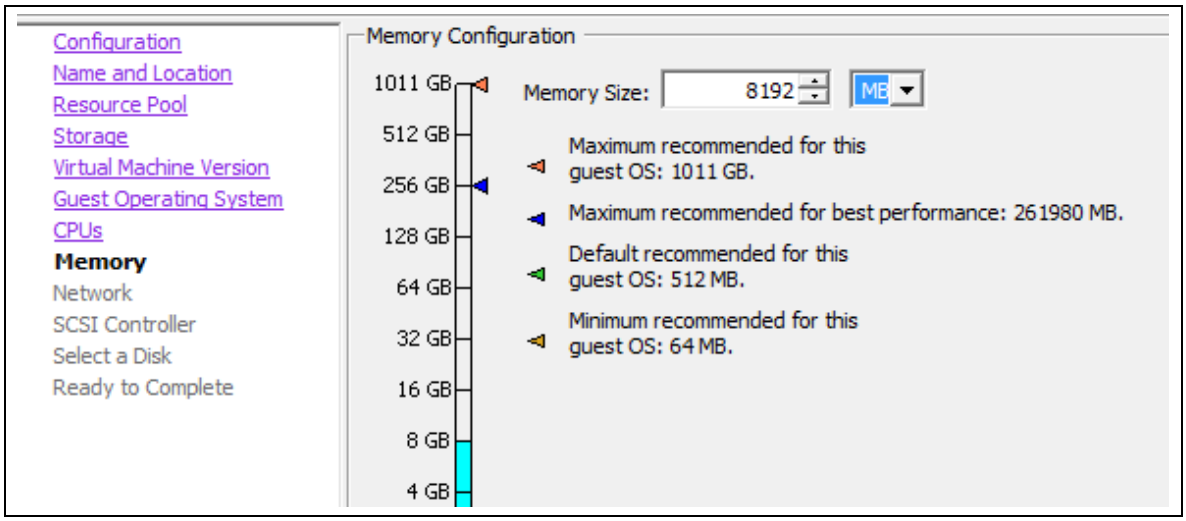


Figure . Assign memory to VFP VM

10. For the network connections option select four virtual interfaces – the first one will be attached to the eth0 of the VFP and the second one to the eth1 interface. The next interfaces will be the data plane interfaces. Therefore, connect the first virtual interface to the bridge br-ext and the second interface to the bridge br-int-vmx1. For both interfaces choose the E1000 adapter. For the third interface, attach it to the bridge vmx1-ge-0/0/0 with a VMXNET3 adapter (this adapter is actually a paravirtualized device and will be attached to the ge-0/0/0 interface of vmx1). Finally add a fourth interface attached to the bridge br-vmx-vmx2 with a E1000 adpater: it will be attached to ge-0/0/1 interface of vmx1.

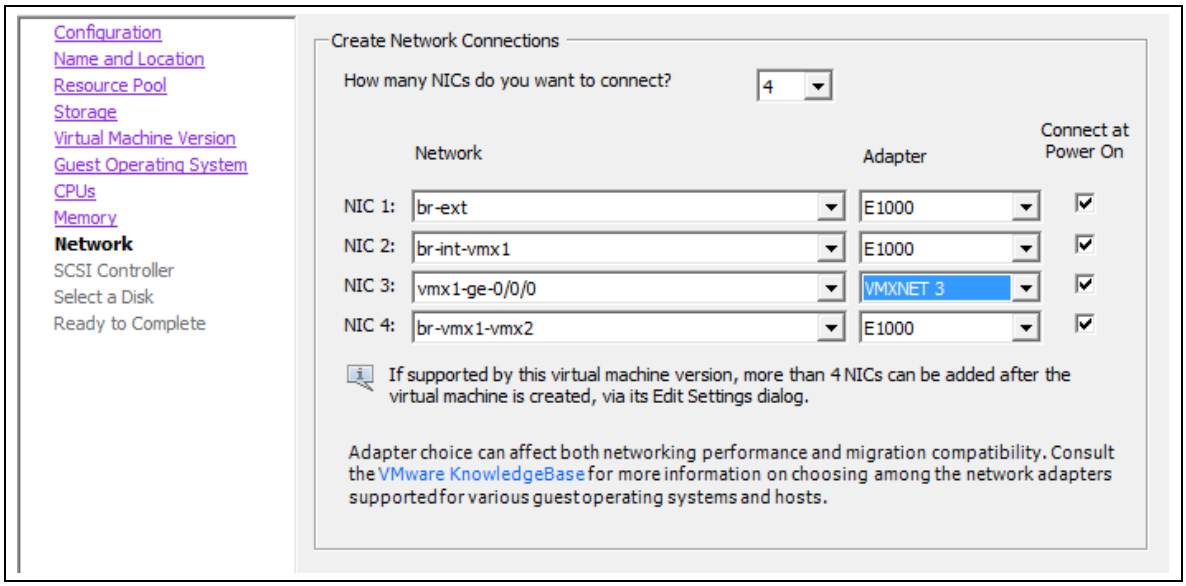


Figure . Configure virtual interfaces of VFP VM

11. Let the LSI Logic Parallel selected as SCSI controller
12. Regarding the Hard Disk select Use Existing virtual Disk.
13. Then choose the following disk into the vmx1 folder: vFPC-20151203.vmdk

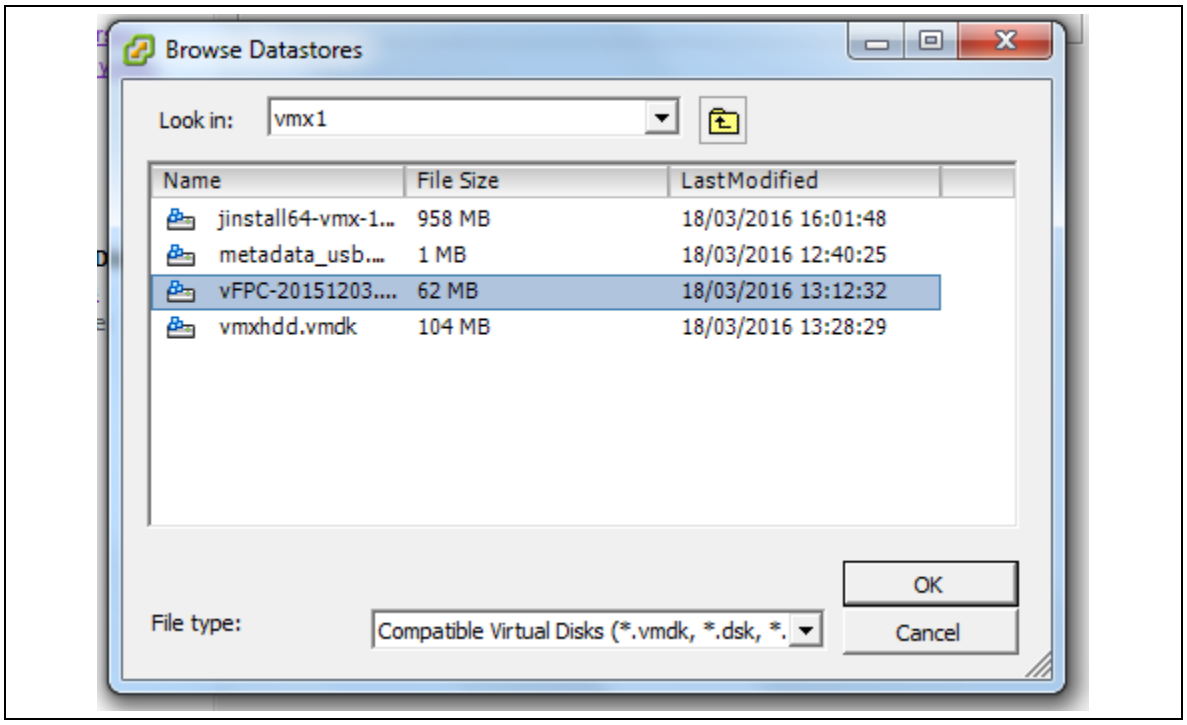


Figure . Add a virtual disk to VFP VM

14. Then finish the installation of the virtual machine

Console port of the VMX router

The two virtual machines VCP and VFP part of the vmx1router are now installed. Before starting the virtual machines you could add a serial port to your virtual machine as followed:

1. Right click on the VCP virtual machine and choose Edit Settings

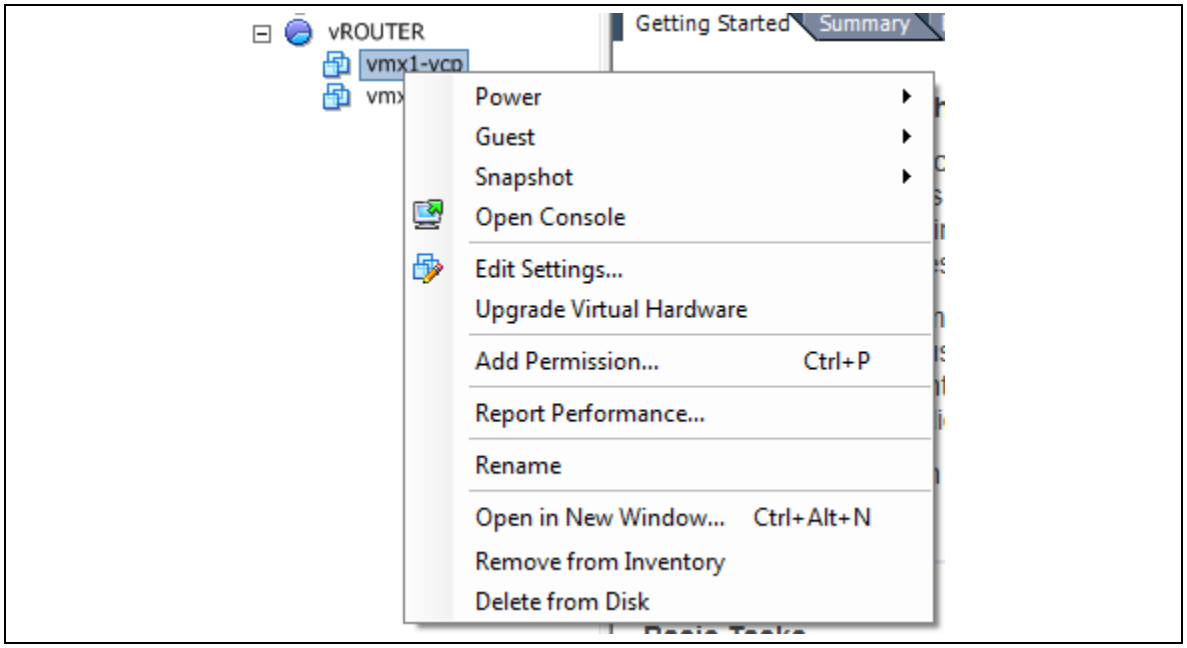


Figure . Edit VCP VM properties

2. Click to the Add button, select Serial Port then press Next and choose Connect via Network

3. Configure the serial access as followed. Hereafter the console port uses the TCP Port 10000. To connect to the console port of the VCP VM of vmx1 you have just to telnet the management IP address of your server on port 10000.

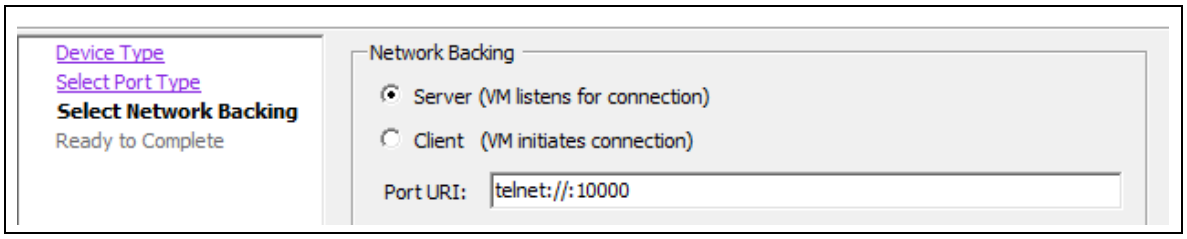


Figure . Add a virtual network serial port

4. You should add a Firewall rules to allow telnet access. Just move to the Configuration tab then on Security Profile option click to Properties

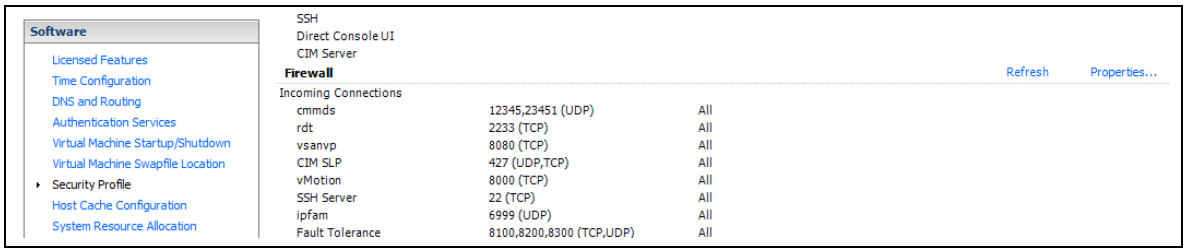


Figure . Access to the security properties of the ESXi

5. Finally enable the option: VM Serial port connected over network

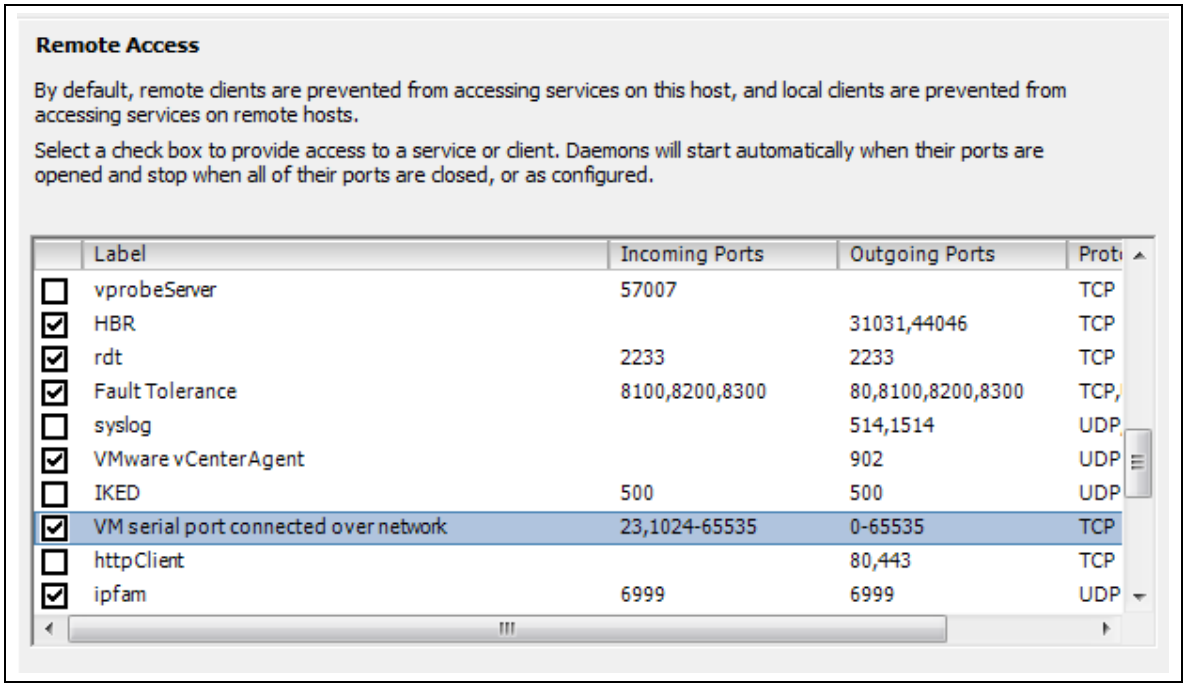


Figure . Allow network serial console port traffic

Initial configuration of VMX

Now let's power on the two virtual machines (VCP and VFP) of vmx1:

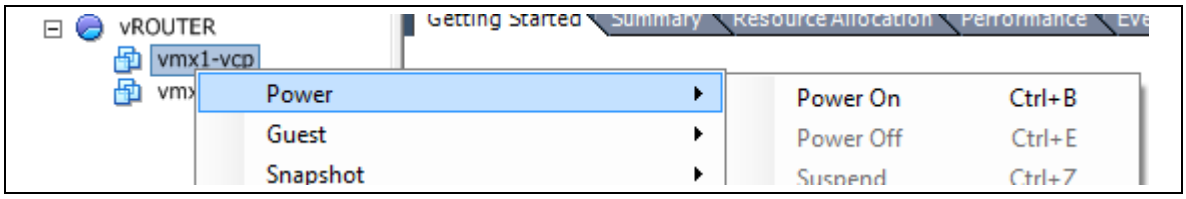


Figure . Power on VCP and VFP VMs

After few minutes you should have access to the console port of the VCP. The default user is root with no password. Then enter in cli mode like that:

```
Amnesiac (ttyd0)

login: root

--- JUNOS 15.1F4.15 built 2015-12-23 20:22:39 UTC
root@% cli
root>
```

You should first see that one FPC is detected:

```
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           Absent   0          0        0      0      0      0         0           0

root> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                VMX755c           VMX
Midplane
Routing Engine 0                                         RE-VMX
CB 0                                                     VMX SCB
CB 1                                                     VMX SCB
FPC 0                                                    Virtual FPC
  CPU            Rev. 1.0 RIOT         123XYZ987
```

After adding the license you should do now some initial configurations. As seen below, we configure the FPC in slot 0 with one PIC made of 8 ports. As of Junos 15.1 only FPC 0 and PIC 0 have a meaning. The number of ports currently supported is 1 up to 23. Even if we only need two ports we allocate 8 GE ports for illustration purposes. The second command is actually a pure cosmetic knob. It allows you to choose the prefix of the VFP's interfaces. You have three choices: `ge`, `xe` or `et`. We select `ge` as our physical port is a 1GE interface: but just for fun we could use `et` and it will work also. We finally add some configuration lines: the hostname, the root password (mandatory), a new user "lab" and we configure the out-of-band management interface fxp0 (attached to the br-ext bridge):

```
[edit]
root# set chassis fpc 0 pic 0 number-of-ports 8

root# set chassis fpc 0 pic 0 interface-type ?
Possible completions:
  et                   Prefix interfaces as et
  ge                   Prefix interfaces as ge
  xe                   Prefix interfaces as xe
[edit]
root# set chassis fpc 0 pic 0 interface-type ge

[edit]
root# set system host-name vmx1

root# set system root-authentication plain-text-password
New password:
Retype new password:
```

```
root# set system login user lab authentication plain-text-password
New password:
Retype new password:

[edit]
root# set system login user lab class super-user

[edit]
root# set interfaces fxp0 unit 0 family inet address 192.168.1.2/24
```

Once the following configuration committed you can check interfaces status:

```
root@vmx1> show interfaces terse | match ge-
ge-0/0/0                up    up
ge-0/0/1                up    up
ge-0/0/2                up    down
ge-0/0/3                up    down
ge-0/0/4                up    down
ge-0/0/5                up    down
ge-0/0/6                up    down
ge-0/0/7                up    down
```

As observed, only two interfaces are UP as we installed a VMX with only two data plane interfaces: the first one ge-0/0/0 is connected to vmnic2 through the virtual bridge vmx1-ge-0/0/0 and the second one is ge-0/0/1, a pure virtual interface connected to the virtual switch br-vmx1-vmx2. The initial configuration is finished.

Finalize your lab topology

To finalize our topology we need to create another VMX: vmx2. Just repeat all the above steps to create the VCP and VFP VMs of vmx2. We only summarize below the steps that you should take care:

- Copy the four files of the VMX package installation into a new folder of the datastore: vmx2. You could upload the files one more time or just copy/paste existing files of the vmx1 folder to vmx2 folder.
- For VCP make sure you configure the two virtual interfaces like that. Remember that the first interface is always the management interface and the second one the internal interface uses for VCP and VFP communication:

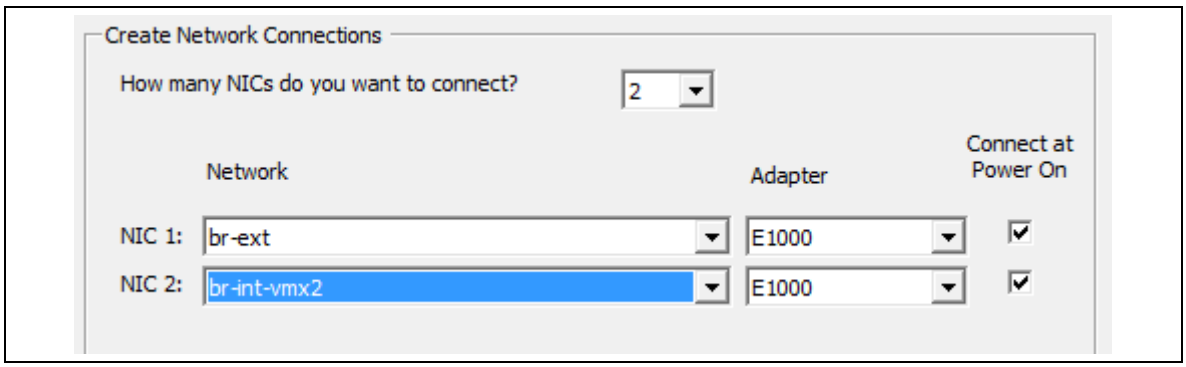


Figure . Virtual interfaces of VCP of the second VMX router

- For VFP make sure you configure four virtual interfaces as followed. The first two interfaces are used respectively for management and internal traffic. The next interfaces are data plane interfaces attached to ge-0/0/x interfaces of the VMX:

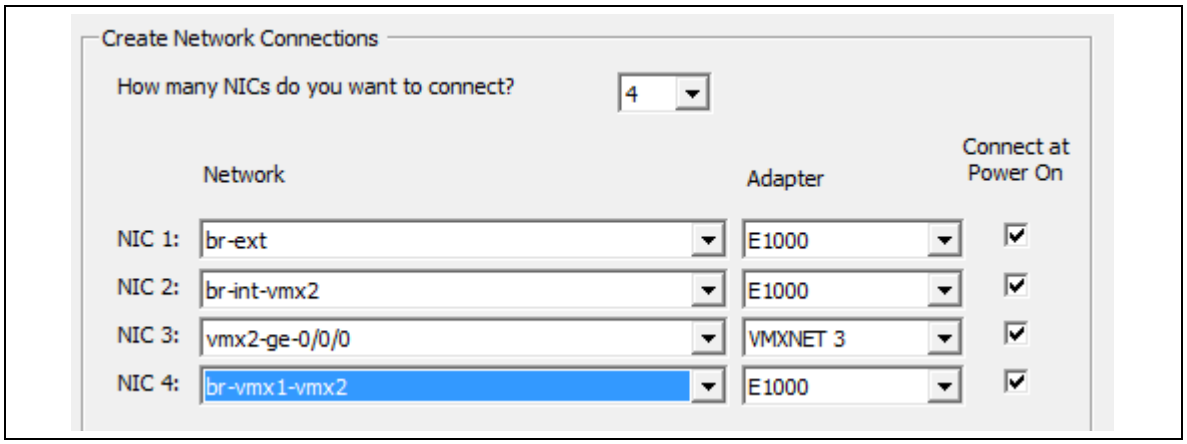


Figure . Virtual interfaces of VFP of the second VMX router

- Add a serial port to the VCP VM of vmx2 and finally carry out the initial configuration of the vmx2.

Installing VMX for low-bandwidth applications

The previous installation part covered a very simple way to deploy a VMX router using the GUI of the vSphere Client. It shown with VMware you can install and interconnect very quickly several VMX for lab simulation.

During the next part we will focus on the installation of VMX for low-bandwidth applications. Even if we can use ESXi with Paravirtualization interfaces (commonly name vmxnet3 on VMware) to achieve that, we decided to switch to a new hypervisor, Linux/KVM.

The aim is to deploy on a single server the following topology depicted by the Figure :

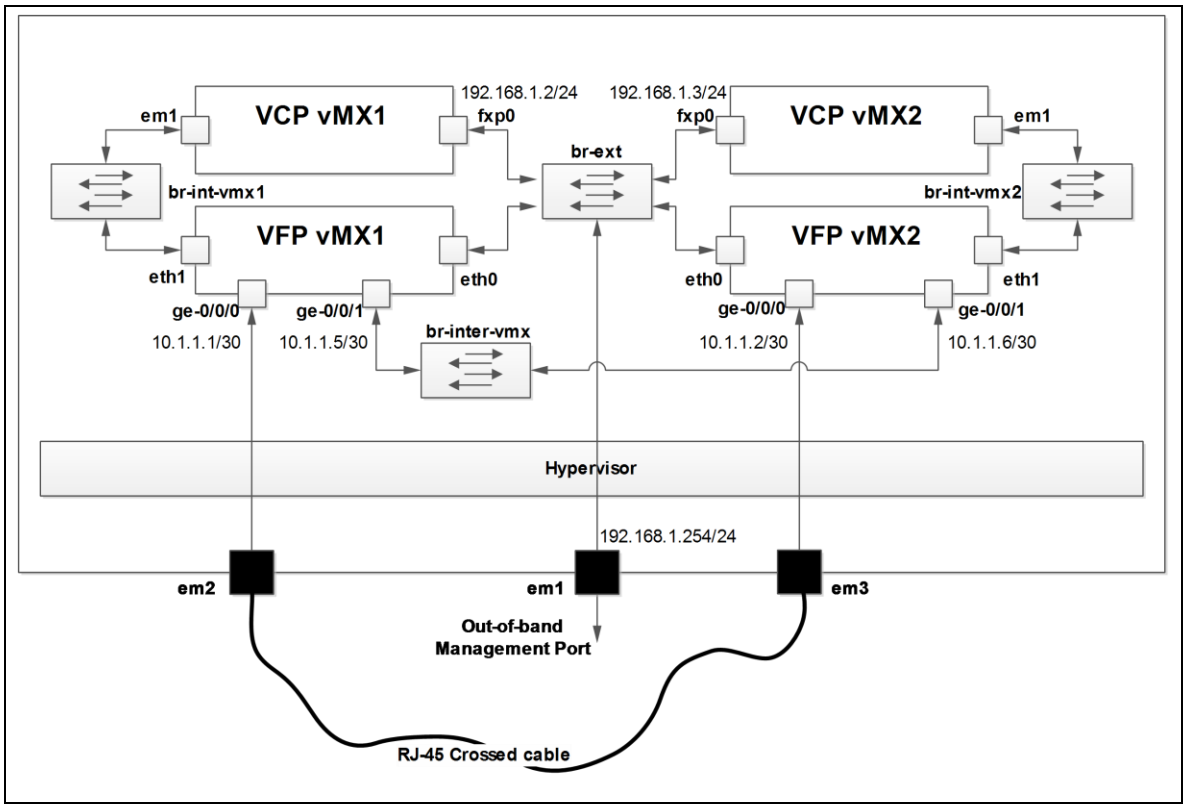


Figure . Two VMX configured on KVM

As seen, each VMX router has got two interfaces. One connected to a 1GE physical port of the server, respectively em2 for vmx1 and em3 for VMX2. The second interfaces of both VMX are internally crossed connected together through a virtual bridge. IP addressing of VFP interfaces and for management interface (fxp0) of both VMX are also described on the figure above.

Server and host OS pre-requires

There are only few hardware and software requirements for low-bandwidth applications which are:

- Processor has to support VT-X. All recent x86 (Intel or AMD) processors support today standard Virtualization Technique). This following command provides you the information if VT-X is supported on your CPU:

```
jnpr@kvm:~$ lscpu | grep Virtualization
Virtualization:        VT-x
```

- Make sure your server has enough memory and cpu capacities to install at least one VMX instance. For that refer to the table X-X.
- The virtio interface support is required for better network I/O performances.

Host OS and KVM installation

As of Junos 15.1F4 the supported host Operating System for VMX over Linux/KVM is Ubuntu 14.04 LTS. You can download online the ISO file of the Ubuntu 14.04 LTS on many repositories. Make sure you choose the "Server" distribution. Then simply build a bootable USB key based on this ISO and start the installation of Ubuntu on your server. This chapter does not cover the full Ubuntu installation. Nevertheless at the installation's step "Software Selection" make sure you select the "Virtual Machine Host" package. This one includes KVM/QEMU software. Hereafter, we select also OpenSSH server to access our server through a secure connection.

```
[*] OpenSSH server
[ ] DNS server
```

```
[ ] LAMP server
[ ] Mail server
[ ] PostgreSQL database
[ ] Print server
[ ] Samba file server
[ ] Tomcat Java server
[*] Virtual Machine host
[*] Manual package selection
```

Once your server is installed, just check you have the recommended Ubuntu version:

```
jnpr@kvm:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.4 LTS
Release:        14.04
Codename:       trusty
```

And also check the version of KVM which should be at least 2.0.0:

```
jnpr@kvm:~$ kvm --version
QEMU emulator version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1.22), Copyright (c) 2003-2008 Fabrice
Bellard
```

The server uses in this lab hosts four 1GE interfaces configured as followed. The em1 is the management interface used to reach the server itself and also the management interface of both VMX routers. Remember em1 port will attached to a virtual bridge that will also connect the fxp0 interfaces of VCP instances.

```
jnpr@kvm:~$ more /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto em1
iface em1 inet static
        address 192.168.1.254
        netmask 255.255.255.0
        network 193.168.1.0
        broadcast 193.168.1.255
        gateway 193.168.1.253
        dns-nameservers 192.168.36.1

auto em2
iface em2 inet static
        address 10.0.0.1
        netmask 255.255.255.0

auto em3
iface em3 inet static
        address 10.0.1.1
        netmask 255.255.255.0

auto em4
iface em4 inet static
        address 10.0.2.1
        netmask 255.255.255.0
```

Make sure your network configuration allows you to access to Internet. Moreover if you use an http proxy you should add the following configurations in order to download the recommended packages. Hereafter how to set up your http proxy respectively for `apt` and `wget` commands:

```
jnpr@kvm:~$ more /etc/apt/apt.conf
Acquire::http::Proxy "http://<ip-proxy-@>:<port>/";
```

```
jnpr@kvm:~$ more /home/jnpr/.wgetrc
http_proxy = http://<ip-proxy-@>:<port>/
use_proxy = on
wait = 15
```

Packages installation

The next step consists in installing the required packages. For low-application mode there are only few packages to install. Nevertheless some libraries, already installed during the server installation, should be updated to work with VMX. The recommended packages will be installed with the `apt` command. As mentioned apt command requires an Internet connection. First of all, update from the Ubuntu repositories the list of available packages:

```
jnpr@kvm:~$ sudo apt-get update
[…]
Fetched 215 kB in 9s (22.1 kB/s)
Reading package lists... Done
```

Then perform the installation of the recommended packages. You can do it one by one or in one line as followed:

```
jnpr@kvm:~$ sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl python-netifaces
vnc4server libyaml-dev python-yaml libparted0-dev libpciaccess-dev libnuma-dev libyajl-dev
libxml2-dev libglib2.0-dev libnl-dev libnl-dev python-pip python-dev libxml2-dev libxslt-dev
```

Once all packages are installed, you can call back the above command. This is actually the best way to check that all required packages are well installed:

```
jnpr@kvm:~$ sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl python-netifaces
vnc4server libyaml-dev python-yaml libparted0-dev libpciaccess-dev libnuma-dev libyajl-dev
libxml2-dev libglib2.0-dev libnl-dev libnl-dev python-pip python-dev libxml2-dev libxslt-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libxslt1-dev' instead of 'libxslt-dev'
bridge-utils is already the newest version.
libpciaccess-dev is already the newest version.
libxslt1-dev is already the newest version.
libyajl-dev is already the newest version.
python is already the newest version.
python-dev is already the newest version.
python-netifaces is already the newest version.
libnl-dev is already the newest version.
libglib2.0-dev is already the newest version.
libnuma-dev is already the newest version.
libparted0-dev is already the newest version.
libvirt-bin is already the newest version.
libxml2-dev is already the newest version.
libyaml-dev is already the newest version.
python-yaml is already the newest version.
qemu-kvm is already the newest version.
numactl is already the newest version.
python-pip is already the newest version.
vnc4server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
```

Great! The last step is to check the version of libvirt. Libvirt is a collection of open source API, daemon and management tools that provide a convenient way to manage virtual machines and other virtualization functionalities such as storage and network interface management. Libvirt is used to manage KVM, Xen, VMware ESX, QEMU and other popular virtualization technologies. The software components include:

- An API library,
- A daemon (libvirtd), and
- A command line utility (virsh).

To view which version is installed, you can call this command:

```
jnpr@kvm:~$ libvirtd --version
libvirtd (libvirt) 1.2.2
```

The minimum version supported by VMX is libvirt 1.2.8. So you might need to adjust the version by upgrading the libvirt package. First download with the wget command the sources of libvirt in a temporary folder and decompress the package:

```
jnpr@kvm:/var/tmp$ cd /var/tmp/

jnpr@kvm:/var/tmp$ wget http://libvirt.org/sources/libvirt-1.2.8.tar.gz

jnpr@kvm:/var/tmp$ tar zxvf libvirt-1.2.8.tar.gz

jnpr@kvm:/var/tmp$ ls
libvirt-1.2.8  libvirt-1.2.8.tar.gz
```

Then, uninstall the previous version of libvirt:

```
jnpr@kvm:/var/tmp$ cd libvirt-1.2.8

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo ./configure --prefix=/usr/local --with-numactl

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo service libvirt-bin stop

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make uninstall
```

Once carried out, configure and install the libvirt 1.2.8 as followed:

```
jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo ./configure --prefix=/usr --localstatedir=/ --with-numactl

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make install
```

Finally, start the new version of libvirt and check if the right version is running:

```
jnpr@kvm:~$ sudo service libvirt-bin start

jnpr@kvm:~$ libvirtd --version
libvirtd (libvirt) 1.2.8
```

Sounds good! Your host OS is now ready to begin the installation of the VMX router. You can download the VMX image from the Juniper website and put it on your home folder. Here we download the 15.1F4 installation package:

```
jnpr@kvm:~$ ls /home/jnpr/vmx-15.1F4.15.tgz
vmx-15.1F4.15.tgz
```

Prepare your "work folder" on Linux

VMX installation package contains several files and folders. If you wish to deploy several VMX instances on the same server you should organize your "work folder" to avoid any mistake. We only provide here some recommendations and you can organize your "work folder" as you wish.

Here we created four directories under our root folder /var/vRouters/

```
jnpr@kvm:/var/vRouters$ ls
dev-scripts  junos  vmx1  vmx2
```

dev-scripts

It will contain all the instance of vmx-junosdev.conf file. This file describes how virtual interfaces of vmx instances are connected.

junos

This folder will group all vmx installation packages

vmx1

This folder will contain the specific configuration file of the vmx1 instance. This configuration file is derived from the vmx.conf

vmx2

This folder will contain the specific configuration file of the vmx2 instance. This configuration file is derived from the vmx.conf

> In our case we wish to run two VMX routers this is why we only created two vmx folders. You can create more than two vmx folders if needed.

Now, decompress the VMX package into the junos/ folder (here vmx-15.1F4.15.tgz):

```
jnpr@kvm:~$ sudo tar zxvf /home/jnpr/vmx-15.1F4.15.tgz --directory /var/vRouters/junos/
```

Now let's analyze the content of the decompressed packages:

```
jnpr@kvm:/var/vRouters/junos$ ls vmx-15.1F4-3/
build  config  docs  drivers  env  images  scripts  vmx.sh
```

As you can see there are several directories and one file: vmx.sh which is actually the main script provided by Juniper to deploy and manage a VMX instance. Let's have a look at the VMX package contents:

build

This directory will contain the built vmx instances.

config

This folder contains two templates of configuration. The vmx.conf is used to deploy the VMX VCP and VFP virtual machines. It includes, among others, the name of the VMX instance, the source images of the VCP and VFP VMs, the number of interfaces attached to the VFP. The second template is vmx-junosdev.conf. This file allows creating interfaces binding. In other words how a virtual NIC of the VFP is connected (to a physical interface, to a virtual bridge…).

docs

This folder includes some configuration file examples and documentation files.

drivers

This folder includes the modified source code of the ixgbe driver. This driver is needed for PCI-Passthrough mode which is required by high-bandwidth applications.

env

It contains OS environment settings.

images

This folder includes the software image for VCP (jinstall-vmx*.img), the image file for VCP file storage (vmxhdd.img) and the software image file for VFP (vFPC_*.img).

scripts

It includes all the Juniper Networks orchestration scripts that are called by the main script vmx.sh.

vmx.sh

The main orchestration script.

Understanding the VMX configuration file

The installation package is decompressed and your work folder is ready. Before deploying the first VMX router we need to understand the config/vmx.conf file. This file will be used by the main orchestration script to deploy your VMX. The vmx.conf file is implemented in YAML format: a human friendly language. Some python scripts will then convert this file to several XML files used by libvirt for VM deployment.

> YAML (YAML Ain't Markup) is a human friendly data serialization language. VMX config file uses YAML because it's as close to plain English as data serialization and configuration formats get. The advantage of YAML is that it does not require curly braces, allowing you to omit quotation marks for strings in most cases,

relying on indentation for structure, which makes it much more readable compared to XML. Important tips about YAML: YAML relies on indentation to understand the data structure: use spaces instead of tabs, tabs are not universally supported across implementations. It is also case sensitive. In short, every space/indentation matters. Taking cautions when modifying the vmx.conf file. A good practice is to just replace the parameters (numbers, image path, interface names, MAC, etc) and leave everything else intact.

So simply editing the file and have a look at some important lines:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# more config/vmx.conf
################################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
################################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier               : vmx1   # Maximum 4 characters ⓪
    host-management-interface : eth0 ①
    routing-engine-image      : "/home/vmx/vmxlite/images/jinstall64-vmx.img" ②
    routing-engine-hdd        : "/home/vmx/vmxlite/images/vmxhdd.img" ③
    forwarding-engine-image   : "/home/vmx/vmxlite/images/vPFE.img" ④

---
#External bridge configuration ⑤
BRIDGES:
    - type  : external
      name  : br-ext                 # Max 10 characters

---
#vRE VM parameters ⑥
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 1024
    console_port: 8601

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.94
        macaddr   : "0A:00:DD:C0:DE:0E"

---
#vPFE VM parameters ⑦
FORWARDING_PLANE:
    memory-mb   : 6144
    vcpus       : 3
    console_port: 8602
    device-type : virtio ⑧

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.98
        macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces ⑨
JUNOS_DEVICES:
   - interface           : ge-0/0/0
     mac-address         : "02:06:0A:0E:FF:F0"
```

```
    description          : "ge-0/0/0 interface"

  - interface            : ge-0/0/1
    mac-address          : "02:06:0A:0E:FF:F1"
    description          : "ge-0/0/1 interface"

  - interface            : ge-0/0/2
    mac-address          : "02:06:0A:0E:FF:F2"
    description          : "ge-0/0/2 interface"

  - interface            : ge-0/0/3
    mac-address          : "02:06:0A:0E:FF:F3"
    description          : "ge-0/0/3 interface"
```

Now, let's clarify each line flagged with a number:

- Line (0): the identifier (ID) is the name of your vmx router. The VCP and VFP instances will be named as followed: vcp-<ID> and vfp-<ID>
- Line (1): this is the current management interface of your server. This interface will be attached to the virtual bridge br-ext.
- Line (2): the binary image of the VCP VM.
- Line (3): the binary image of the VCP hard disk.
- Line (4): the binary image of the VFP VM.
- Line (5): refer to the Figure , this is actually the virtual bridge that will interconnect the management port of the server and the management interfaces of the VCP and VFP virtual machines.
- Line (6): this part describes the memory/cpu allocation and management parameters of the VCP virtual machine. The console port is a port number bound to the localhost interface (127.0.0.1) of the server. The IP address provided there is the address of the fxp0 interface.
- Line (7): this part describes the memory/cpu allocation and management parameters of the VFP virtual machine. The console port is a port number bound to the localhost interface of the server. The IP address provided there is the address of the eth0 interface.
- Line (8): this is a VFP specific statement. The device-type will determine which IO virtualization technology for data plane interfaces will be used: it could be `virtio` or `sriov` mode.
- Line (9): this part describes how many virtual NICs are attached to the VFP virtual machine. The naming of each interface has always this syntax: ge-0/0/x - independently of the real bandwidth of the physical port. At this level we don't precise if the virtual NIC is attached to a physical port or to a virtual bridge. The MAC address of each interface should be unique among with all interfaces but also among with all VMX instances running on the same server.

We are finally ready! No more theoretical explanation, let's now deploy your first VMX on KVM.

Deploying a VMX instance with the orchestration script

Based on the config/vmx.conf template and the information provided by the Figure we are going to deploy our first VMX router. The best way to instance a VMX is to duplicate the vmx.conf template in the target router instance folder. In our case we just copy config/vmx.conf to /var/vRouters/vmx1/:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# cp config/vmx.conf ../../vmx1/
```

We have modified the vmx1/vmx.conf as followed in order to match the requirements provided by the Figure :

```
jnpr@kvm:/var/vRouters# more vmx1/vmx.conf
############################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
```

```
#  Leave a space after ":" to specify the parameter value.
#
############################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx1   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      : "/var/vRouters/junos/vmx-15.1F4-3/images/jinstall64-vmx-15.1F4.15-
domestic.img"
    routing-engine-hdd        : "/var/vRouters/junos/vmx-15.1F4-3/images/vmxhdd.img"
    forwarding-engine-image   : "/var/vRouters/junos/vmx-15.1F4-3/images/vFPC-20151203.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 10000

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.2
        macaddr   : "0A:00:DD:00:01:01"

---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 8192
    vcpus       : 3
    console_port: 10001
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.21
        macaddr   : "0A:00:DD:00:01:02"

---
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     mac-address          : "02:06:0A:00:01:01"
     description          : "ge-0/0/0 interface"

   - interface            : ge-0/0/1
     mac-address          : "02:06:0A:00:01:02"
     description          : "ge-0/0/1 interface"
```

The instance will be named vmx1. The three images are located in /var/vRouters/junos/vmx-15.1F4-3/images/ folder. The host-management-interface of our server is the em1 interface. As requested by the table X-X, we allocate 1 vCPU and 2GB of memory for the VCP virtual machine and 3 vCPU and 8GB of memory for the VFP VM. The 192.168.1.2 and 192.168.1.21 are respectively the management IP addresses for the fxp0 interface of VCP VM and eth0 interface of VFP VM. The console port for VCP is 10000 and 10001 for VFP. The virtualization technology uses for VFP network I/O is virtio. We also configure two

virtual NIC attached to VFP: ge-0/0/0 and ge-0/0/1. We will see later how to connect these interfaces to a physical port or a virtual bridge.

One word about MAC addresses: As mentioned, you should configure unique MAC addresses. We use this convention to allocate the MAC addresses:

- For the management interface of VCP and VFP we use the following MAC template: 0A:00:DD:00:XX:YY where XX represents the number of the VMX instance (here above instance number 1) and YY is an incremental counter to allocate a MAC address to VCP and VFP virtual machines. We use here YY=01 for VCP and YY=02 for VFP.
- For data plane interfaces attached to the VFP virtual machine we use the following MAC template: 02:06:0A:00:XX:YY where XX represents the number of the VMX instance (here above instance number 1) and YY is an incremental counter to allocate a MAC address for each virtual NIC attached to the VFP instance. We use here YY=01 for ge-0/0/0 and YY=02 for ge-0/0/1

We will use the `vmx.sh` main orchestration script to deploy the vmx1 configuration. Before deploying our vmx1 router, just call this script without any parameter to show the help. As you see the help is enough explicit and no need to explain more each option:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh

Usage: vmx.sh [CONTROL OPTIONS]
       vmx.sh [LOGGING OPTIONS] [CONTROL OPTIONS]
       vmx.sh [JUNOS-DEV BIND OPTIONS]
       vmx.sh [CONSOLE LOGIN OPTIONS]

    CONTROL OPTIONS:
       --install                      : Install And Start VMX
       --start                        : Start VMX
       --stop                         : Stop VMX
       --restart                      : Restart VMX
       --status                       : Check Status Of VMX
       --cleanup                      : Stop VMX And Cleanup Build Files
       --cfg <file>                   : Override With The Specified vmx.conf File
       --env <file>                   : Override With The Specified Environment .env File
       --build <directory>            : Override With The Specified Directory for Temporary Files
       --help                         : This Menu

    LOGGING OPTIONS:
       -l                             : Enable Logging
       -lv                            : Enable Verbose Logging
       -lvf                           : Enable Foreground Verbose Logging

    JUNOS-DEV BIND OPTIONS:
       --bind-dev                     : Bind Junos Devices
       --unbind-dev                   : Unbind Junos Devices
       --bind-check                   : Check Junos Device Bindings
       --cfg <file>                   : Override With The Specified vmx-junosdev.conf File

    CONSOLE LOGIN OPTIONS:
       --console [vcp|vfp] [vmx_id]   : Login to the Console of VCP/VFP

    VFP Image OPTIONS:
       --vfp-info <VFP Image Path>    : Display Information About The Specified vFP image

Copyright(c) Juniper Networks, 2015
```

We have just highlighted three options on the previous output. Indeed, the aim it's to install (`--install` option) the VMX router based on the specific vmx1/vmx.conf file (`--cfg` option). The last option (`-lv`) just enables the verbose mode and might be useful to troubleshoot installation when this one failed. Let's start! The output has been truncated to avoid too many pages of logs:

```
 jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh -lv --install --cfg ../../vmx1/vmx.conf
==================================================
    Welcome to VMX
```

```
================================================
Date.....................................03/13/16 17:19:49
VMX Identifier.....................................vmx1
Config file......................................./var/vRouters/vmx1/vmx.conf
Build Directory.................................../var/vRouters/junos/vmx-15.1F4-3/build/vmx1
Environment file................................../var/vRouters/junos/vmx-15.1F4-
3/env/ubuntu_virtio.env
Junos Device Type...............................virtio
[…]
================================================
    VMX Status Verification Completed.
================================================
Log file.........................................
    /var/vRouters/junos/vmx-15.1F4-3/build/vmx1/logs/vmx_1457885989.log
================================================
    Thankyou for using VMX
================================================
```

Awesome! Our first VMX router has been successfully installed. Let's check the build folder to verify that the vmx1 is well created:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# ls -li build/
total 4
34865854 drwxr-xr-x 5 root root 4096 Mar 13 17:19 vmx1
```

You could play with the `virsh` command line (provided by the libvirt package) in order to check if the two virtual machines have been installed as expected and know their status:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
```

As seen the two virtual machines are currently running. You can retrieve detailed information such as the number of vCPU or the memory allocated of each virtual machine with another virsh option. Just provide the instance ID as parameter, retrieved with the previous command):

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# virsh dominfo 3
Id:             3
Name:           vfp-vmx1
UUID:           502f910c-8b08-4737-8945-6254a4bc9c1b
OS Type:        hvm
State:          running
CPU(s):         3
CPU time:       24.4s
Max memory:     8000512 KiB
Used memory:    8000000 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: none
Security DOI:   0
```

Now, let's check the status of the virtual NICs and virtual bridges. Indeed, the vmx.sh script has created several virtual NICs and bridges based on the vmx.conf file. The command to manage virtual bridges is `brctl`:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.d89d67767e18       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vfp_ext-vmx1
br-int-vmx1     8000.52540002a198       yes             br-int-vmx1-nic
                                                        vcp_int-vmx1
```

```
                                                        vfp_int-vmx1
virbr0          8000.fe060a000101       yes             ge-0.0.0-vmx1
                                                        ge-0.0.1-vmx1
```

First of all, as you can see, it is very simple to retrieve which interface or bridge is attached to which VMX instance. Indeed, each virtual network item is suffixed with the name of the VMX: here the vmx1.

The orchestration script has created two management interfaces, one for VCP and another one for VFP. These interfaces are respectively named: vcp_ext-vmx1 and vfp_ext-vmx1. These two interfaces are actually mapped to fxp0 and eth0 interfaces. These interfaces are attached, with the management interface of the server itself (em1), to a virtual bridge br-ext. This one will provide you the ability to access to your VCP and VFP virtual machines through the out-of-band management network. The script also created two other internal interfaces vcp_int-vmx1 and vfp_int-vmx1. The two interfaces refer to the em1 and eth1 interfaces of the VCP and VFP virtual machines. The two internal interfaces are connected together via a dedicated virtual bridge named br-int-vmx1. Finally, we find back our two data plane interfaces ge-0/0/0 and ge-0/0/1 respectively identified as ge-0.0.0-vmx1 and ge-0.0.1-vmx1. These two interfaces are attached to the bridge virbr0. What does it mean? The virbr0 is the default bridge. As mentioned the binding of data plane interfaces is not performed during the installation. This will be done during the next step. Thus, by waiting their binding, the data plane interfaces are attached to the default bridge.

You can also retrieve some interesting information related to the virtual interfaces with the `ip` command:

```
jnpr@kvm:/home/iptac# sudo ip link | grep vmx1
20: br-int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
21: br-int-vmx1-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master br-int-vmx1 state DOWN
mode DEFAULT group default qlen 500
22: vcp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br-ext state
UNKNOWN mode DEFAULT group default qlen 500
23: vcp_int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br-int-vmx1
state UNKNOWN mode DEFAULT group default qlen 500
24: vfp_ext-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br-ext state
UNKNOWN mode DEFAULT group default qlen 500
25: vfp_int-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br-int-vmx1
state UNKNOWN mode DEFAULT group default qlen 500
26: ge-0.0.0-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
UNKNOWN mode DEFAULT group default qlen 500
27: ge-0.0.1-vmx1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
UNKNOWN mode DEFAULT group default qlen 500
```

Binding physical devices to VFP

The final step to achieve our vmx1 installation is to bind the data plane interfaces to the right physical port or virtual bridge. Based on the Figure the ge-0/0/0 should be attached to the physical port em2 and the ge-0/0/1 to a virtual bridge named br-inter-vmx. These tasks are one more time carried out through the main orchestration script vmx.sh. This time, we use another configuration file and options. To proceed to interface binding you must first edit the following file:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# ls config/vmx-junosdev.conf
config/vmx-junosdev.conf
```

As we did with the vmx.conf template, it is recommend to create a copy of this default file. We had decided to place the interface binding specific files in the dev-script/ folder. This is done as followed:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# cp config/vmx-junosdev.conf ../../dev-scripts/vmx1-
dev.conf
```

Now, let's edit the vmx1-dev.conf file. Remember it is currently just a copy of the original file:

```
jnpr@kvm:/var/vRouters# more dev-scripts/vmx1-dev.conf
###############################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
```

```
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
############################################################
interfaces :

     - link_name  : vmx_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

     - link_name  : vmx_link2
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

     - link_name  : vmx_link3
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/1
       endpoint_2 :
         - type        : host_dev
           dev_name    : eth3

     - link_name  : vmx_link4
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/2
       endpoint_2 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/2
```

This file uses also the YAML language so take caution with indentation and YAML restrictions.

A device binding is identified uniquely by a `link_name`. Each `link_name` is made of two endpoints. There are currently three types of endpoint:

- `junos_dev`: a virtual NIC attached to a given VMX. A `junos_dev` endpoint requires to specify `vm_name` – the VMX unique name declared in the vmx.conf file during the installation – and the `dev_name` – a data plane interface attached to the VFP; interface also declared in the vmx.conf file.
- `bridge_dev`: the endpoint is a virtual bridge. If the bridge is not present the orchestration script will create it before.
- `host_dev`: the endpoint is a physical port of the server.

Based on these information we could easily deduced with connections would be established if we try to use the default template file:

- For link vmx_link1: the main script will connect the ge-0/0/0 interface of the VMX vmx1 to the virtual bridge bridge1.
- For link vmx_link2: the script will attach the ge-0/0/0 interface of the VMX vmx2 to the virtual bridge bridge1.
- For link vmx_link3: the main script will connect the ge-0/0/1 interface of the VMX vmx1 to the physical port eth3.
- For link vmx_link4: the script will attach the ge-0/0/2 interface of the VMX vmx1 to the interface ge-0/0/2 of the VMX vmx2. This is a direct link which simulates a virtual crossed cable.

The following figure illustrated this sample topology:

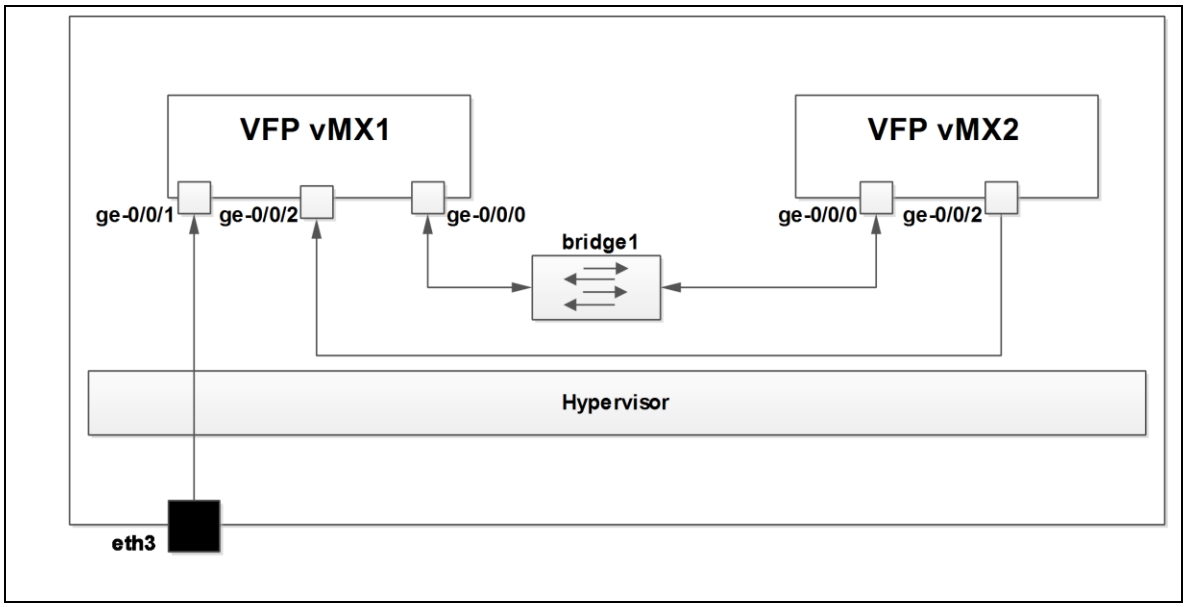


Figure . The default interface binding configuration.

Now, let's modify the vmx1-dev.conf in order to match our requirements:

```
jnpr@kvm:/var/vRouters/dev-scripts# more vmx1-dev.conf
################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
################################################################
interfaces :

     - link_name  : vmx1_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
```

```
    endpoint_2 :
      - type        : host_dev
        dev_name    : em2

  - link_name   : vmx1_link2
    mtu         : 1500
    endpoint_1 :
      - type        : junos_dev
        vm_name     : vmx1
        dev_name    : ge-0/0/1
    endpoint_2 :
      - type        : bridge_dev
        dev_name    : br-inter-vmx
```

We want here to create two links: the first one to attach the ge-0/0/0 interface to the physical port em2 and the second one to connect the ge-0/0/1 interface to the virtual bridge br-inter-vmx. The next step is to use the orchestration script to make the binding based on this above configuration file:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --bind-dev --cfg ../../dev-scripts/vmx1-dev.conf
Checking package ethtool.........................[OK]
Bind Link vmx1_link1(ge-0.0.0-vmx1, em2).........[OK]
Numa node for em2................................0
Cores servicing numa node 0......................0-3
Pid of vfp-vmx1..................................41402
Pin vhost-41402 (PID=41406) to cores 0-3.........[OK]
Pin vhost-41402 (PID=41405) to cores 0-3.........[OK]
Pin vhost-41402 (PID=41404) to cores 0-3.........[OK]
Pin vhost-41402 (PID=41403) to cores 0-3.........[OK]
Bind Bridge port br-inter-vmx(ge-0.0.1-vmx1)......[OK]
```

All seems to have worked as expected. To double check the interfaces binding you should use one more time the orchestration script with another option:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --bind-check --cfg ../../dev-scripts/vmx1-dev.conf
Checking package ethtool.........................[OK]
Check Link vmx1_link1(ge-0.0.0-vmx1, em2).........[OK]
Check Bridge port br-inter-vmx(ge-0.0.1-vmx1).....[OK]
```

Great! Another way is to use the `brclt` command. Remember, before performing the interfaces binding, the data plane interfaces of vmx1 were attached to the default bridge virbr0. Now, the em2 and ge-0/0/0 interface are linked together and ge-0/0/1 is connected to the new bridge br-int-vmx1:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.d89d67767e18       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vfp_ext-vmx1
br-int-vmx1     8000.52540079f281       yes             br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1
br-inter-vmx    8000.fe060a000102       no              ge-0.0.1-vmx1
virbr0          8000.000000000000       yes
vmx1_link1      8000.d89d67767e19       no              em2
                                                        ge-0.0.0-vmx1
```

All is done for vmx1. It's time to access to the VMX and perform the initial configurations and checks.

Access to VMX and initial configuration of VMX

The first access to your VMX router is done through the virtual console port. Remember you specified a console port number for both VCP and VFP virtual machines. The ports were respectively 10000 and 10001 for VCP and VFP. There are two methods to access to your VMX via its console port:

- By using the orchestration script as followed. Here you must specify for which VMX instance and which virtual machine (VCP or VFP) you want to access:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --console vcp vmx1
--
Login Console Port For vcp-vmx1 - 10000
Press Ctrl-] to exit anytime
--
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.


Amnesiac (ttyd0)

login:
```

- Or simply use the telnet command targeting the localhost and the configured console port. Hereafter to access to VCP:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# telnet localhost 10000
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.


Amnesiac (ttyd0)

login:
```

> To exist the console mode just press Ctrl^] keys.

The default user is root with no password. Then enter in cli mode like that:

```
Amnesiac (ttyd0)

login: root

--- JUNOS 15.1F4.15 built 2015-12-23 20:22:39 UTC
root@% cli
root>
```

You should first see that one FPC is detected:

```
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           Absent   0          0        0      0      0      0         0          0

root> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                VMX755c           VMX
Midplane
Routing Engine 0                                         RE-VMX
CB 0                                                     VMX SCB
CB 1                                                     VMX SCB
FPC 0                                                    Virtual FPC
  CPU            Rev. 1.0 RIOT         123XYZ987
```

After adding the license you should do now some initial configurations as we did for VMX on ESXi:

```
[edit]
root# set chassis fpc 0 pic 0 number-of-ports 8
```

```
root# set chassis fpc 0 pic 0 interface-type ?
Possible completions:
  et                   Prefix interfaces as et
  ge                   Prefix interfaces as ge
  xe                   Prefix interfaces as xe
[edit]
root# set chassis fpc 0 pic 0 interface-type ge

[edit]
root# set system host-name vmx1

root# set system root-authentication plain-text-password
New password:
Retype new password:

root# set system login user lab authentication plain-text-password
New password:
Retype new password:

[edit]
root# set system login user lab class super-user

[edit]
root# set interfaces fxp0 unit 0 family inet address 192.168.1.2/24
```

Once the following configuration committed you can check interfaces status:

```
root@vmx1> show interfaces terse | match ge-
ge-0/0/0                up    up
ge-0/0/1                up    up
ge-0/0/2                up    down
ge-0/0/3                up    down
ge-0/0/4                up    down
ge-0/0/5                up    down
ge-0/0/6                up    down
ge-0/0/7                up    down
```

As observed, only two interfaces are UP as we installed a VMX with only two data plane interfaces. The initial configuration is finished. You can now play with the vmx1 as a classical MX. The following steps will allow you to create a second VMX and interconnect it to the vmx1.

Interconnect two VMX instances

We are now installing a new VMX router named vmx2. We will move faster as all steps have been described in detail for vmx1. First we create a new configuration file derived from the default template:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# cp config/vmx.conf ../../vmx2/
```

The configuration file of vmx2 is the following:

```
jnpr@kvm:/var/vRouters/vmx2# more vmx.conf
############################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
############################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier              : vmx2   # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image    : "/var/vRouters/junos/vmx-15.1F4-3/images/jinstal
164-vmx-15.1F4.15-domestic.img"
```

```
    routing-engine-hdd        : "/var/vRouters/junos/vmx-15.1F4-3/images/vmxhdd.
img"
    forwarding-engine-image   : "/var/vRouters/junos/vmx-15.1F4-3/images/vFPC-20
151203.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 20000

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.3
        macaddr   : "0A:00:DD:00:02:01"

---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 8192
    vcpus       : 3
    console_port: 20001
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.22
        macaddr   : "0A:00:DD:00:02:02"

---
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     mac-address          : "02:06:0A:00:02:01"
     description          : "ge-0/0/0 interface"

   - interface            : ge-0/0/1
     mac-address          : "02:06:0A:00:02:02"
     description          : "ge-0/0/1 interface"
```

As shown, we modified the identifier, the management IP addresses and console ports and the MAC addresses that should be unique. Then, we proceed to the deployment of this second VMX instance:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh -lv --install --cfg ../../vmx2/vmx.conf
============================================
    Welcome to VMX
============================================
Date.....................................03/15/16 17:11:15
VMX Identifier.....................vmx2
[…]
============================================
    VMX Status Verification Completed.
============================================
Log file.....................................
    /var/vRouters/junos/vmx-15.1F4-3/build/vmx2/logs/vmx_1458058275.log
============================================
    Thankyou for using VMX
```

```
================================================
```

Installation has been done with success. Let's check how many virtual machines are currently running:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
 5     vcp-vmx2                       running
 6     vfp-vmx2                       running
```

As expected, there are four virtual machines which run. The next step consists in performing the interfaces binding of the vmx2. For that we derive from the file vmx-junosdev.conf this following file:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# cp config/vmx-junosdev.conf ../../dev-scripts/vmx2-dev.conf
```

Once modified the file vmx2-dev.conf looks like that:

```
jnpr@kvm:/var/vRouters/dev-scripts# more vmx2-dev.conf
################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
################################################################
interfaces :

     - link_name  : vmx2_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : host_dev
           dev_name    : em3

     - link_name  : vmx2_link2
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/1
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : br-inter-vmx
```

As required by the Figure , the ge-0/0/0 interface of vmx2 is attached to the physical port em3 and the interface ge-0/0/1 to the virtual bridge `br-inter-vmx`. Let's proceed to the interfaces binding:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --bind-dev --cfg ../../dev-scripts/vmx2-dev.conf
Checking package ethtool.........................[OK]
Bind Link vmx2_link1(ge-0.0.0-vmx2, em3).........[OK]
Numa node for em3................................0
Cores servicing numa node 0......................0-3
Pid of vfp-vmx2..................................1875
```

```
Pin vhost-1875 (PID=1879) to cores 0-3............[OK]
Pin vhost-1875 (PID=1878) to cores 0-3............[OK]
Pin vhost-1875 (PID=1877) to cores 0-3............[OK]
Pin vhost-1875 (PID=1876) to cores 0-3............[OK]
Bind Bridge port br-inter-vmx(ge-0.0.1-vmx2)......[OK]
```

You can check the status of the interfaces binding for vmx2 and finally have a look at bridge status to validate that there is no more interfaces attached to the default bridge virbr0:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --bind-check --cfg ../../dev-
scripts/vmx2-dev.conf
Checking package ethtool..........................[OK]
Check Link vmx2_link1(ge-0.0.0-vmx2, em3).........[OK]
Check Bridge port br-inter-vmx(ge-0.0.1-vmx2).....[OK]

jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.d89d67767e18       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vcp_ext-vmx2
                                                        vfp_ext-vmx1
                                                        vfp_ext-vmx2
br-int-vmx1     8000.525400f2abb7       yes             br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1
br-int-vmx2     8000.525400f7bd13       yes             br-int-vmx2-nic
                                                        vcp_int-vmx2
                                                        vfp_int-vmx2
br-inter-vmx    8000.fe060a000102       no              ge-0.0.1-vmx1
                                                        ge-0.0.1-vmx2
virbr0          8000.000000000000       yes
vmx1_link1      8000.d89d67767e19       no              em2
                                                        ge-0.0.0-vmx1
vmx2_link1      8000.d89d67767e1a       no              em3
                                                        ge-0.0.0-vmx2
```

As shown the two VMX routers are now internally connected together through the virtual bridge br-inter-vmx. The interface ge-0/0/0 of both VMX is attached to a physical port of the server. We access to our vmx2 via its console port and fill the same initial configuration as the vmx1:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --console vcp vmx2
login: root

--- JUNOS 15.1F4.15 built 2015-12-23 20:22:39 UTC
root@% cli
root> edit
Entering configuration mode

[edit]
root# set chassis fpc 0 pic 0 number-of-ports 8

[edit]
root# set chassis fpc 0 pic 0 interface-type ge

[edit]
root# set system host-name vmx2

[edit]
root# set system root-authentication plain-text-password

[edit]
root# set system login user lab authentication plain-text-password

[edit]
```

```
root# set system login user lab class super-user

[edit]
root# set interfaces fxp0 unit 0 family inet address 192.168.1.3/24
```

To finalize our setup and based on the addressing of the Figure we will try to establish two OSPF adjacencies. The configuration of vmx1 and vmx2 is derived from this configuration – only IP addresses are different:

```
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 10.1.1.x/30;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 10.1.1.x/30;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 172.16.20.x/32;
            }
        }
    }
}
protocols {
    ospf {
        area 0.0.0.0 {
            interface ge-0/0/0.0 {
                interface-type p2p;
            }
            interface ge-0/0/1.0 {
                interface-type p2p;
            }
            interface lo0.0 {
                passive;
            }
        }
    }
}
```

Once committed on both VMX we can finally check the OSPF neighbors. Remember that physical ports em2 and em3 are connected to a crossed cable:

```
root@vmx1> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.1.1.2         ge-0/0/0.0             Full      172.16.20.2      128    39
10.1.1.6         ge-0/0/1.0             Full      172.16.20.2      128    32
```

Awesome, isn't it? Our two VMX for low-bandwidth applications are now running and both pure virtual interfaces and physical interfaces are operational as shown by the status of two OSPF adjacencies.

Start / Stop / Restart / Remove a VMX instance

You can easily start, stop or restart a VMX instance by using the orchestration script. To stop a given VMX router just proceed as followed:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --stop --cfg ../../vmx2/vmx.conf
```

Don't forget to provide the configuration file of the given VMX router in parameter.

To start or restart a VMX router let's do like that:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --start --cfg ../../vmx2/vmx.conf

jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --restart --cfg ../../vmx2/vmx.conf
```

Finally you can remove a VMX instance which includes:

- Removing the virtual machines associated: the VCP and VFP
- Removing virtual interfaces

Here, we want to remove vmx. For that we use the cleanup option:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --cleanup --cfg ../../vmx2/vmx.conf
====================================================
    Welcome to VMX
====================================================
Date.....................................................03/15/16 17:54:34
VMX Identifier.....................................vmx2
[…]
====================================================
    VMX Stop Completed
====================================================
Cleanup auto-generated files.....................[OK]
====================================================
    VMX Cleanup Completed
====================================================
Log file.........................................../dev/null
====================================================
    Thankyou for using VMX
====================================================
```

You can check the current running virtual machines:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
```

vmx2 associated virtual machines have been removed. To finish the cleaning process, just unbind the interfaces of the vmx2:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh --unbind-dev --cfg ../../dev-
scripts/vmx2-dev.conf
Checking package ethtool.........................[OK]
Unbind Link vmx2_link1(ge-0.0.0-vmx2, em3).......[OK]
Unbind Bridge port br-inter-vmx(ge-0.0.1-vmx2)....[OK]
```

Don't forget to provide the right device file of the given VMX router in parameter.

The brctl command shows you that there is no more interface and links related to the vmx2 router:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.d89d67767e18       yes             br-ext-nic
                                                        em1
                                                        vcp_ext-vmx1
                                                        vfp_ext-vmx1
br-int-vmx1     8000.525400f2abb7       yes             br-int-vmx1-nic
                                                        vcp_int-vmx1
                                                        vfp_int-vmx1
br-inter-vmx    8000.fe060a000102       no              ge-0.0.1-vmx1
virbr0          8000.000000000000       yes
vmx1_link1      8000.d89d67767e19       no              em2
```

```
                                                    ge-0.0.0-vmx1
```

Installing VMX for high-bandwidth applications

In this part we will focus on the installation of VMX for high-bandwidth applications. When a use case requires more than 3Gbps of traffic you have to switch to a specific configuration of the VMX. Moreover some specifics hardware capabilities become mandatory and some system adjustments will be required. As of Junos 15.1, performance mode for high-bandwidth applications is only supported on Linux/KVM.

Server and host OS requirements

There are some hardware and software requirements for high-bandwidth applications which are:

- Processor has to support VT-d/IOMMU feature.
- Make sure your server supports SR-IOV. This feature must be enabled. This is a BIOS setting example:

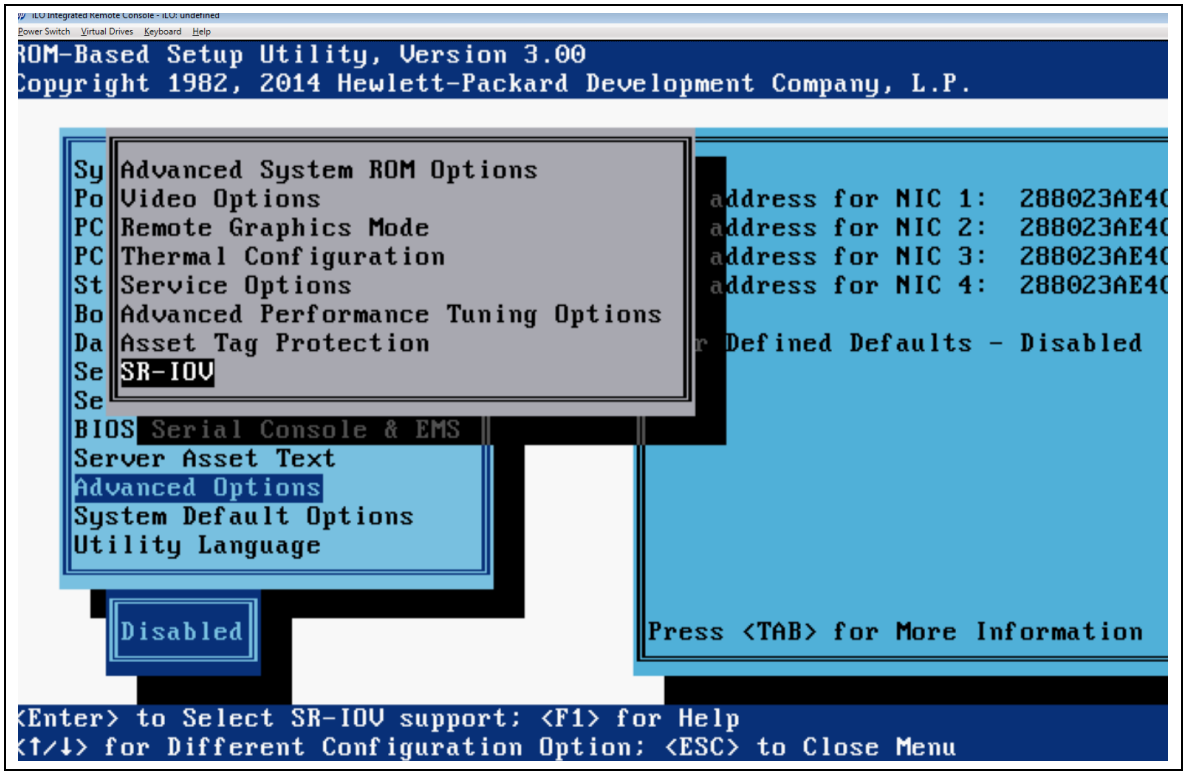


Figure . SR-IOV BIOS configuration.

- The interface module must use Intel 82599 10GE controller with Intel x520, x540 or x560 adapters.
- VT-d/IOMMU feature must be enabled on your host OS.
- For multicast support and IXGBE driver compilation: the host OS Kernel must use the version 3.13.0-32
- For multicast support: The 10GE NIC driver should use the Juniper ixgbe driver.
- The HyperThreading feature is recommended for better performances at high rate. This feature should be enabled at the BIOS level:

  HyperThreading: This feature allows a single physical core to behave as two logical cores. In this configuration a core can execute two threads at the same time. Notice that this does not double the performance. The VMX release requires HyperThreading to be enabled to support the flowcache feature. This HyperThreading function is not mandatory and is checked by the orchestration script during the installation procedure.

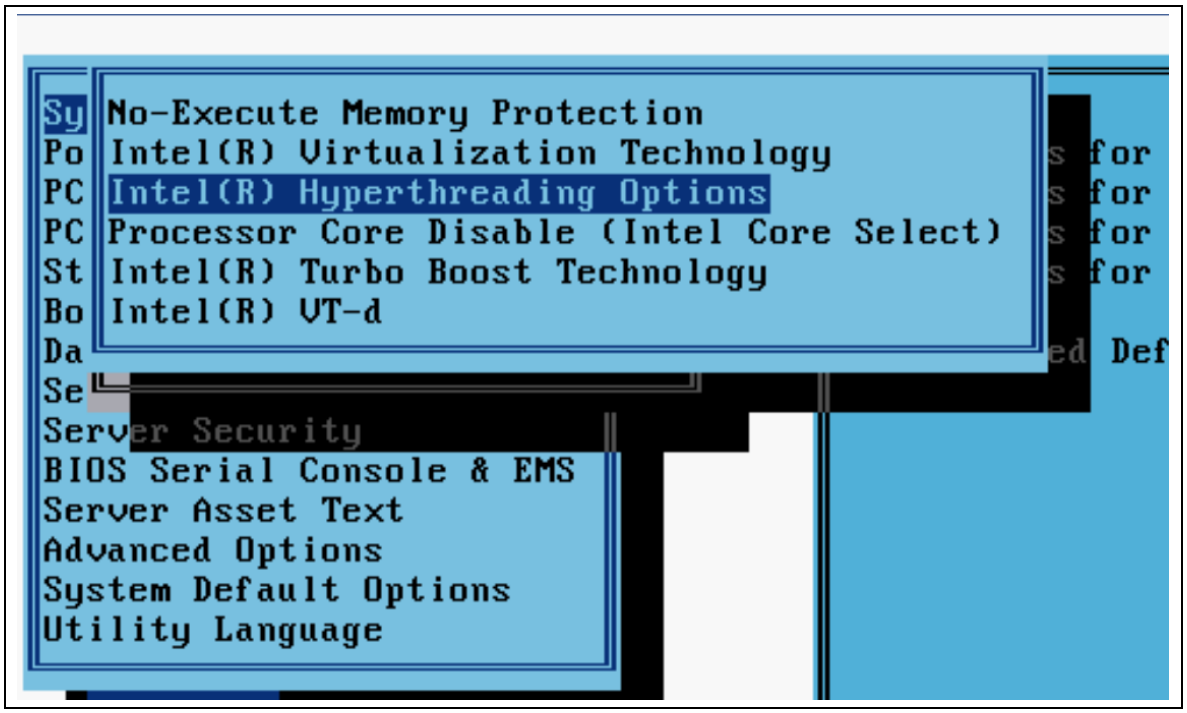


Figure . HyperThreading option in the BIOS.

Host OS and KVM installation

The first step is to adjust your BIOS setting by enabling VT-d/IOMMU, SR-IOV and if supported the HyperThreading functions. Then, you have to install the Ubuntu 14.04 LTS as host OS. As for low-bandwidth applications, make sure you choose the "Server" distribution and you select the "Virtual Machine Host" package during the installation procedure.

Once your Ubuntu OS is installed you need first to install and enable a specific Linux Kernel. Currently the ixgbe driver coming with Ubuntu does not work with virtual routers. The main issue is lack of multicast support on ingress - packet received on a Virtual Function (VF) will be discarded silently and won't be delivered into the guest VM, an example of the immediate effect of this is that OSPF (and most of today's IGP) neighborship won't come up. Therefore building VMX based on SR-IOV requires compiling the ixgbe kernel driver from source code, which is provided by Juniper to fix the multicast support. The code is available in the installation package.

> At the time of writing the book there is problem to compile ixgbe from source code under any kernels other than 3.13.0-32-generic. That's why the kernel needs to be changed in this setup. Nevertheless, if you don't need to handle multicast traffic (control and data planes) you can bypass the kernel upgrade step and ixgbe recompilation.

No worries! Changing a Linux Kernel is not complex anymore. You have to simply download and install the required Kernel with the `apt` command:

```
jnpr@kvm:~$ sudo sudo apt-get install linux-firmware linux-image-3.13.0.32-generic linux-image-
extra-3.13.0.32-generic
```

Let's then modify the file /boot/grub/grub.cfg. Indeed, to force grub to boot first on the Linux Kernel 3.13.0-32 you have to move on top of the list the `menuentry` referring to Linux 3.12.0-32. Actually just move the block of config just after this line: `export linux_gfx_mode`

For our server this looks like as followed:

```
jnpr@kvm:~$ more /boot/grub/grub.cfg
[…]
else
```

```
  set linux_gfx_mode=text
fi
export linux_gfx_mode
menuentry 'Ubuntu, with Linux 3.13.0-32-generic (recovery mode)' --class ubuntu --class gnu-linux
--class gnu --class os $menuentry_id_option 'gnulinu
x-3.13.0-32-generic-recovery-f97821ef-f320-4341-a7dc-00a656b2afc9' {
                recordfail
                load_video
                insmod gzio
                insmod part_msdos
                insmod ext2
                set root='hd0,msdos1'
                if [ x$feature_platform_search_hint = xy ]; then
                  search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-
efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  f97821ef-f320-4341-a7dc-
00a656b2afc9
                else
                  search --no-floppy --fs-uuid --set=root f97821ef-f320-4341-a7dc-00a656b2afc9
                fi
                echo    'Loading Linux 3.13.0-32-generic ...'
                linux   /boot/vmlinuz-3.13.0-32-generic root=UUID=f97821ef-f320-4341-a7dc-
00a656b2afc9 ro recovery nomodeset
                echo    'Loading initial ramdisk ...'
                initrd  /boot/initrd.img-3.13.0-32-generic
        }
```

Once your Linux Kernel is upgraded you have to enable VT-d/IOMMU on your host OS kernel. This is simply done like that. Just add a configuration line:

```
jnpr@kvm:~# sudo echo 'GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on pci=realloc"' >>
/etc/default/grub
```

And then check that the line has been added:

```
jnpr@kvm:/home/iptac# grep -i iommu /etc/default/grub
GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on pci=realloc"
```

Then update grub as followed:

```
jnpr@kvm:~# sudo update-grub
```

Finally reboot your server. Now let's do some checks. First check the version of your kernel which should be 3.13.0-32:

```
jnpr@kvm:~$ uname -a
Linux kvm 3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014 x86_64 x86_64 x86_64
GNU/Linux
```

You could check that IOMMU has been well enabled by looking at parameters passed to the kernel at the time it is started. You should have the `intel_iommu` option passed and set to "on":

```
jnpr@kvm:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-3.13.0-32-generic root=UUID=f97821ef-f320-4341-a7dc-00a656b2afc9 ro
intel_iommu=on pci=realloc
```

Packages installation

The next step consists in installing the required packages. As for low-application mode there are some required packages to install. First of all, update from the Ubuntu repositories the list of available packages:

```
jnpr@kvm:~$ sudo apt-get update
[…]
Fetched 215 kB in 9s (22.1 kB/s)
Reading package lists... Done
```

Then perform the installation of the recommended packages. You can do it one by one or in one line (as followed):

```
jnpr@kvm:~$ sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl python-netifaces
vnc4server libyaml-dev python-yaml libparted0-dev libpciaccess-dev libnuma-dev libyajl-dev
libxml2-dev libglib2.0-dev libnl-dev libnl-dev python-pip python-dev libxml2-dev libxslt-dev
```

Once all packages are installed, you can call back the above command. This is actually the best way to check that all required packages are well installed:

```
jnpr@kvm:~$ sudo apt-get install bridge-utils qemu-kvm libvirt-bin python numactl python-netifaces
vnc4server libyaml-dev python-yaml libparted0-dev libpciaccess-dev libnuma-dev libyajl-dev
libxml2-dev libglib2.0-dev libnl-dev libnl-dev python-pip python-dev libxml2-dev libxslt-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libxslt1-dev' instead of 'libxslt-dev'
bridge-utils is already the newest version.
libpciaccess-dev is already the newest version.
libxslt1-dev is already the newest version.
libyajl-dev is already the newest version.
python is already the newest version.
python-dev is already the newest version.
python-netifaces is already the newest version.
libnl-dev is already the newest version.
libglib2.0-dev is already the newest version.
libnuma-dev is already the newest version.
libparted0-dev is already the newest version.
libvirt-bin is already the newest version.
libxml2-dev is already the newest version.
libyaml-dev is already the newest version.
python-yaml is already the newest version.
qemu-kvm is already the newest version.
numactl is already the newest version.
python-pip is already the newest version.
vnc4server is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
```

As for low-application check your libvirt version and update it if needed. To view the current version of libvirt installed:

```
jnpr@kvm:~$ libvirtd --version
libvirtd (libvirt) 1.2.2
```

The minimum version supported by VMX is libvirt 1.2.8. So you might need to adjust the version by upgrading the libvirt package. First download with the wget command the sources of libvirt in a temporary folder and decompress the package:

```
jnpr@kvm:/var/tmp$ cd /var/tmp/

jnpr@kvm:/var/tmp$ wget http://libvirt.org/sources/libvirt-1.2.8.tar.gz

jnpr@kvm:/var/tmp$ tar zxvf libvirt-1.2.8.tar.gz

jnpr@kvm:/var/tmp$ ls
libvirt-1.2.8  libvirt-1.2.8.tar.gz
```

Then, uninstall the previous version of libvirt:

```
jnpr@kvm:/var/tmp$ cd libvirt-1.2.8

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo ./configure --prefix=/usr/local --with-numactl

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo service libvirt-bin stop

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make uninstall
```

Once carried out, configure and install the libvirt 1.2.8 as followed:

```
jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo ./configure --prefix=/usr --localstatedir=/ --with-numactl

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make

jnpr@kvm:/var/tmp/libvirt-1.2.8$ sudo make install
```

Finally, start the new version of libvirt and check if the right version is installed:

```
jnpr@kvm:~$ sudo service libvirt-bin start

jnpr@kvm:~$ libvirtd --version
libvirtd (libvirt) 1.2.8
```

You can download the VMX image from the Juniper web site and put it on your home folder. Here we download the 15.1F4 installation package:

```
jnpr@kvm:~$ ls /home/jnpr/vmx-15.1F4.15.tgz
vmx-15.1F4.15.tgz
```

> We will organize our "work folder" as for a low-bandwidth applications use case. Please, refer to this paragraph for more information.

The 10GE NIC driver for VMX

In our server we have installed an Intel 82599 10GE controller with a 2x10GE ports HP 560M adapter. As shown below we still have our four 1GE ports but also two new 10GE ports, named p2p1 and p2p2 as displayed by `ifconfig` command:

```
root@kvm:~# ifconfig -a
em1       Link encap:Ethernet  HWaddr d8:9d:67:76:7e:18
          inet6 addr: fe80::da9d:67ff:fe76:7e18/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:258229 errors:0 dropped:0 overruns:0 frame:0
          TX packets:530921 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:109584191 (109.5 MB)  TX bytes:71042844 (71.0 MB)

em2       Link encap:Ethernet  HWaddr d8:9d:67:76:7e:19
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::da9d:67ff:fe76:7e19/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:101 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16298 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9358 (9.3 KB)  TX bytes:1466276 (1.4 MB)

em3       Link encap:Ethernet  HWaddr d8:9d:67:76:7e:1a
          inet addr:10.0.1.1  Bcast:10.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::da9d:67ff:fe76:7e1a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:127 errors:0 dropped:0 overruns:0 frame:0
          TX packets:117 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11362 (11.3 KB)  TX bytes:10598 (10.5 KB)

em4       Link encap:Ethernet  HWaddr d8:9d:67:76:7e:1b
          inet addr:10.0.2.1  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

p2p1      Link encap:Ethernet HWaddr 38:ea:a7:17:65:a0
          inet6 addr: fe80::3aea:a7ff:fe17:65a0/64 Scope:Link
          UP BROADCAST RUNNING PROMISC ALLMULTI MULTICAST MTU:2000 Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:810 (810.0 B) TX bytes:558 (558.0 B)

p2p2      Link encap:Ethernet HWaddr 38:ea:a7:17:65:84
          inet6 addr: fe80::3aea:a7ff:fe17:6584/64 Scope:Link
```

```
          UP BROADCAST RUNNING PROMISC ALLMULTI MULTICAST MTU:2000 Metric:1
          RX packets:1 errors:0 dropped:1 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:301 (301.0 B) TX bytes:558 (558.0 B)
```

To retrieve more information regarding the Ethernet adapters you should use the `lspci` command. First list all your Ethernet PCI devices like that:

```
jnpr@kvm:~# lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.2 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.3 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane Connection
(rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane Connection
(rev 01)
```

The PCI addresses for the two 10GE ports are respectively 06:00.0 and 06:00.1. For more information about a specific 10GE port you should use the following command with specifying a given PCI address:

```
jnpr@kvm:~# sudo lspci -vs 06:00.0
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane
Connection (rev 01)
Subsystem: Hewlett-Packard Company Ethernet 10Gb 2-port 560M Adapter
Physical Slot: 2
Flags: bus master, fast devsel, latency 0, IRQ 136
Memory at eff00000 (32-bit, non-prefetchable) [size=1M]
I/O ports at 6000 [size=32]
Memory at efef0000 (32-bit, non-prefetchable) [size=16K]
[virtual] Expansion ROM at efc00000 [disabled] [size=512K]
Capabilities: [40] Power Management version 3
Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+
Capabilities: [70] MSI-X: Enable+ Count=64 Masked-
Capabilities: [a0] Express Endpoint, MSI 00
Capabilities: [e0] Vital Product Data
Capabilities: [100] Advanced Error Reporting
Capabilities: [140] Device Serial Number 00-00-00-ff-ff-00-00-00
Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
Kernel driver in use: ixgbe
```

The last two lines provide some interesting information. Indeed, the module supports SR-IOV which allows splitting a physical NIC in virtual NIC named Virtual Function (VF). It is interesting to note that by default Virtual Function is disabled. This Intel model adapter currently supports up to 64 VF (aka. 64 virtual NICs) numbered from 0 to 63.

> Please note that currently the VMX orchestration script enables by default one VF (VF number 0) per 10GE NIC. The script actually restarts the ixgbe driver by modifying the `max_vfs` option. In other words the 10GE NIC bandwidth could not be shared between several VMXs.

The last line shows which driver the 10GE port uses. This is, as expected, the ixgbe driver – the one provided by Intel. As mentioned previously, at the time we are writing the book, the current Intel ixgbe driver does not handle ingress multicast traffic. This limitation is for us a drawback because we wish to use OSPF on our VMX. So in our case, we must use the ixgbe driver provided by Juniper.

The source code of the Juniper ixgbe driver can be found into the VMX installation package (vmx-15.1F4-3/drivers/ixgbe-3.19.1/src/):

```
jnpr@kvm:~# cd /var/vRouters/junos/vmx-15.1F4-3/drivers/ixgbe-3.19.1/src/
```

You could check your current driver version by calling this command:

```
jnpr@kvm:~# sudo modinfo ixgbe | grep ver
```

```
version:        4.0.1-k
srcversion:     44CBFE422F8BAD726E61653
vermagic:       3.19.0-25-generic SMP mod_unload modversions
```

The target version is 3.19.1. Here, you see that the Intel driver is not the right one. So let's recompile the Juniper ixgbe driver on your environment:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3/drivers/ixgbe-3.19.1/src# rm -f ixgbe.ko

jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3/drivers/ixgbe-3.19.1/src# make install
make -C /lib/modules/3.13.0-32-generic/build SUBDIRS=/var/vRouters/junos/vmx-15.1F4-
3/drivers/ixgbe-3.19.1/src modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-32-generic'
[...]
ixgbe.
```

Now, compare the compiled driver to the right folder:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3/drivers/ixgbe-3.19.1/src# cmp ixgbe.ko
/lib/modules/3.13.0-32-generic/kernel/drivers/net/ethernet/intel/ixgbe/ixgbe.ko
```

And finally stop and restart the ixgbe driver:

```
jnpr@kvm:~# sudo rmmod ixgbevf

jnpr@kvm:~# sudo rmmod ixgbe

jnpr@kvm:~# sudo modprobe ixgbe
```

> You should encounter an error when stopping the ixgbevf driver: `rmmod: ERROR: Module ixgbevf is not currently loaded`. This driver manages the Virtual Function. As the VF is by default disabled, the ixgbevf might not be started. Don't take into account the error. For more information regarding the option of the ixgbe driver you can read the README file saved here: drivers/ixgbe-3.19.1/

You can call back the `modinfo` command and check back if the driver currently running is the Juniper modified version:

```
jnpr@kvm:~# sudo modinfo ixgbe | grep version
version:        3.19.1
srcversion:     B97B1E7CF79A25F5E4D7B96
vermagic:       3.13.0-32-generic SMP mod_unload modversions
```

Understanding the VMX configuration file for SR-IOV

The server is ready to install the VMX with SR-IOV support. The deployment of an SR-IOV VMX is simply performed by the orchestration script. There are few parameters in the vmx.conf file that differ from the virtio mode. There is a sample config file available in the installation package: config/samples/vmx.conf.sriov. We have extracted below the parts that are specific to the SR-IOV mode:

```
root@kvm:/var/vRouters/junos/vmx-15.1F4-3# more config/samples/vmx.conf.sriov
[…]
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 16384 ❶
    vcpus       : 7 ❷
    console_port: 8602
    device-type : sriov ❸

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.98
        macaddr   : "0A:00:DD:C0:DE:10"

---
```

```
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     port-speed-mbps      : 10000 ❹
     nic                  : int1 ❺
     mtu                  : 2000 ❻            # DO NOT EDIT
     virtual-function     : 0 ❼
     mac-address          : "02:06:0A:0E:FF:F1"
     description          : "ge-0/0/0 connects to int1"

   - interface            : ge-0/0/1
     port-speed-mbps      : 10000
     nic                  : int2
     mtu                  : 2000             # DO NOT EDIT
     virtual-function     : 0
     mac-address          : "02:06:0A:0E:FF:F2"
     description          : "ge-0/0/0 connects to int2"
```

Let's clarify each line flagged with a number:

- Line (1): For full performance mode you need to allocate at least 12GB of memory. Here we allocates 16GB for VFP.
- Line (2): The full performance mode requires 7 vCPU for VFP.
- Line (3): The device type is configured as `sriov` which forces all virtual interfaces of the VCP to be attached to physical NIC via the PCI-Passthrough mechanism.
- Line (4): You must specify the port speed of the NIC. Currently only 10GE SR-IOV NIC are supported.
- Line (5): Specify the NIC name referring to the physical interface name as displayed for example by the ifconfig command.
- Line (6): The MTU is set to 2000 by default
- Line (7): This parameter refers to the VF directly attached to the virtual interface. Currently the orchestration script creates one VF per physical NIC, therefore, VF is always 0.

Deploying a SR-IOV VMX instance via VMX Script

This new VMX router will be named vmx3 and built with two 10GE interfaces as shown by the next figure:

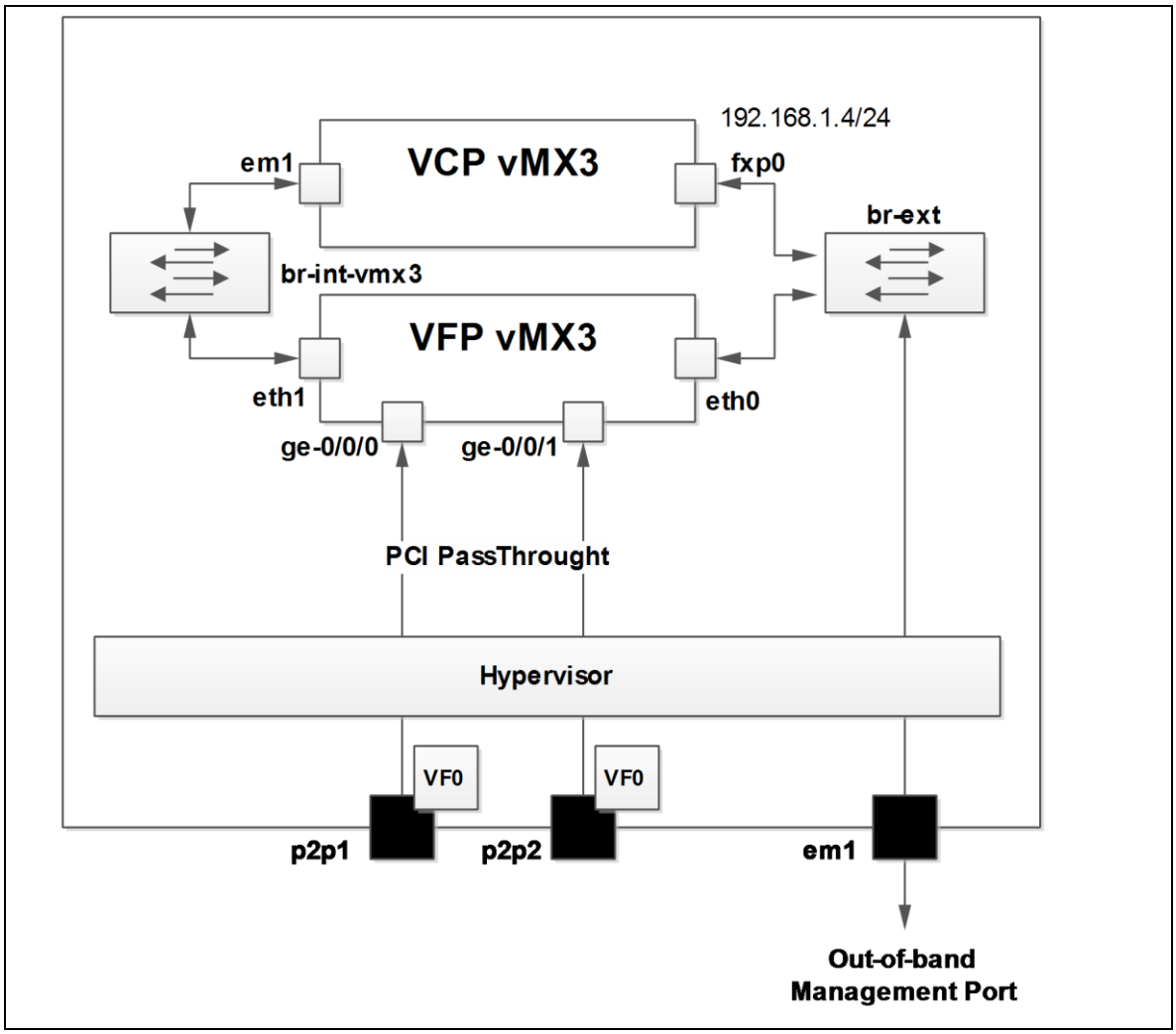


Figure . VMX with PCI-Passthrough enabled

We create a new folder vmx3 and copy the vmx.conf.sriov template;

```
root@kvm:/var/vRouters# mkdir vmx3

root@kvm:/var/vRouters# cp junos/vmx-15.1F4-3/config/samples/vmx.conf.sriov vmx3/vmx-sriov.conf
```

We modify the configuration file as followed:

```
root@kvm:/var/vRouters/vmx3# more vmx-sriov.conf
############################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
############################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier              : vmx3    # Maximum 4 characters
    host-management-interface : em1
    routing-engine-image      : "/var/vRouters/junos/vmx-15.1F4-3/images/jinstall64-vmx-15.1F4.15-
domestic.img"
```

```
    routing-engine-hdd        : "/var/vRouters/junos/vmx-15.1F4-3/images/vmxhdd.img"
    forwarding-engine-image   : "/var/vRouters/junos/vmx-15.1F4-3/images/vFPC-20151203.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                  # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 30000

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.3
        macaddr   : "0A:00:DD:00:03:01"

---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 16384
    vcpus       : 7
    console_port: 30001
    device-type : sriov

    interfaces  :
      - type      : static
        ipaddr    : 192.168.1.24
        macaddr   : "0A:00:DD:00:03:02"

---
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     port-speed-mbps      : 10000
     nic                  : p2p1
     mtu                  : 2000             # DO NOT EDIT
     virtual-function     : 0
     mac-address          : "02:06:0A:00:03:01"
     description          : "ge-0/0/0 connects to p2p1"

   - interface            : ge-0/0/1
     port-speed-mbps      : 10000
     nic                  : p2p2
     mtu                  : 2000             # DO NOT EDIT
     virtual-function     : 0
     mac-address          : "02:06:0A:00:03:02"
     description          : "ge-0/0/1 connects to p2p2"
```

Finally we use the orchestration script to install the vmx3 (note: the output has been truncated):

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo ./vmx.sh -lv --install --cfg ../../vmx3/ vmx-
sriov.conf
================================================
Date.............................................03/13/16 18:19:49
VMX Identifier...................................vmx3
Config file....................................../var/vRouters/vmx3/vmx-sriov.conf
Build Directory................................../var/vRouters/junos/vmx-15.1F4-3/build/vmx3
Environment file................................./var/vRouters/junos/vmx-15.1F4-
3/env/ubuntu_sriov.env
```

```
Junos Device Type.................................sriov
Initialize scripts................................[OK]
Copy images to build directory....................[OK]
================================================
VMX Environment Setup Completed
================================================
[…]
Number of Intel 82599 NICs.......................2
Configuring Intel 82599 Adapters for SRIOV........[OK]
Number of Virtual Functions created..............[OK]
[…]
================================================
48
VMX Status Verification Completed.
================================================
Log file.........................................../dev/null
================================================
Thankyou for using VMX
================================================
```

Sounds good! Let's check with the `virsh` command the status of our VM:

```
jnpr@kvm:/var/vRouters/junos/vmx-15.1F4-3# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 2     vcp-vmx1                       running
 3     vfp-vmx1                       running
 5     vcp-vmx2                       running
 6     vfp-vmx2                       running
 5     vcp-vmx3                       running
 6     vfp-vmx3                       running
```

As shown, we have 3 VMX running. Two VMX run in `virtio` mode and the last one in `sriov` mode. In `sriov` mode there is no need to use orchestration script to bind virtual interfaces to physical interfaces. Remember in the vmx.conf file we have specified the link between the VFP virtual interfaces and the physical NIC. You could call back the `lspci` command to see that the orchestration script has created one Virtual Function per 10GE adapter. These VFs (VF 0) are directly attached to the ge-0/0/0 and ge-0/0/1 interfaces.

```
jnpr@kvm:~# lspci | grep -i ethernet
02:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.2 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
02:00.3 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)
06:00.0 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane Connection
(rev 01)
06:00.1 Ethernet controller: Intel Corporation 82599 10 Gigabit Dual Port Backplane Connection
(rev 01)
06:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
06:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
```

Now your vmx3 will be fully operational after you put the initial configuration via the console port of the VCP, as we did for low-bandwidth applications use case.

Summary of installation procedures

We covered the three typical use cases of VMX. As seen the choice of which Hypervisor depends on your target use case. VMware ESXi is really simple and easy to use for deploying VMX for Lab simulation purposes. KVM is currently the host OS on which you can do the most of tunings and on which you can already set up a VMX supporting several 10Gbps of traffic. The support of Direct I/O on ESXi is in the roadmap and should be available in the coming next releases. The VMX is still at the beginning of the virtual routers era. Many improvements regarding the performance of the VMX as well as the installation procedures should be available soon.