OpenStack: OpenStack Neutron Mellanox ML2 Driver Configuration in Bright

Abstract

This article describes how to use the Mellanox Neutron ML2 Mechanism Driver to add Mellanox InfiniBand support to an existing standard OpenStack Icehouse cloud deployment. The cloud deployment is assumed to be running with ML2 + LinuxBridge Mechanism Driver + VLAN-based network isolation, and is managed with Bright Cluster Manager 7.0.

The end result is:

- extending the capabilities of the OpenStack private cloud with the ability to create OpenStack networks backed with isolated segments of the InfiniBand fabric,
- and the ability to then spawn OpenStack VMs which have direct access to those networks/segments via a dedicated (passthrough) virtual IB device (exposed via SR-IOV).
- such a VM will have
    - an IPoIB network device,
    - direct access to the IB fabric segment (e.g. for running MPI jobs with other machines attached to this segment),
    - also, optionally regular virtual Ethernet devices connected to a VLAN-backed OpenStack networks.
- Users will be able to pick whether they want to create
    - VMs attached to the InfiniBand-backed networks (IPoIB),
    - VLAN/VXLAN-backed isolated Ethernet networks,
    - Flat (shared) cluster internal networks,
    - or any combination of those.

This article focuses on enabling IB functionality alongside the pre-existing regular VLAN-based network isolation. However, it's also possible to follow most of this document as a guide to configuring IB functionality for OpenStack deployments running VXLAN-based network isolation. Some tips on that are included in the text.

Mellanox ML2 Mechanism Driver Introduction

The Mellanox ML2 Mechanism Driver supports Mellanox embedded switch functionality as part of the VPI (Ethernet/InfiniBand) HCA. The Mellanox ML2 Mechanism Driver provides functional

parity with the Mellanox Neutron plugin.

It supports DIRECT (PCI passthrough) and MACVTAP (virtual interface with a tap-like software interface) vnic types. For vnic type configuration API details, please refer to the configuration reference guide at
http://docs.openstack.org/api/openstack-network/2.0/content/binding_ext_ports.html

Hardware vNICs mapped to the guest VMs allow higher performance and advanced features such as RDMA (remote direct memory access).

The driver supports the VLAN network type, so that virtual networks can be supported on Ethernet or on InfiniBand fabrics. In such configurations:

- Mellanox Openstack Neutron Agent (L2 Agent) runs on each compute node.
- The L2 Agent should apply VIF connectivity based on mapping between a VIF (VM vNIC) and Embedded Switch port.

Source: https://wiki.openstack.org/wiki/Mellanox-Neutron-ML2

Prerequisites

- Mellanox ConnectX-3 InfiniBand cards
- A Functioning OpenStack Icehouse private cloud deployed and managed with Bright Cluster Manager 7.0. The deployment should have been deployed with support for Bright-managed instances (i.e. you should have at least one virtual node typically given the name "vnode001", in your cluster's configuration).
- A Headnode CMDaemon revision equal to or higher than r22282 is recommended. Using an older CMDaemon version is also possible, but changes in some OpenStack config files will have to be introduced manually, by freezing the file using cmd.conf, rather than by setting the "configfilemodifications" field in the OpenStackNode role. *More specifically, this applies only to changes in the sections of config files which contain "_" in the section name, e.g. [linux_bridge] (Tip: You can run "cmd -v" on the headnode to check the revision).*
- Basic proficiency in using Bright CMSH

- VT-d and SR-IOV enabled in the BIOS (VT-d may sometimes appear under "Virtualization Technology" in the BIOS processor settings, which also includes enabling VT-x for KVM etc.)

For those of you who are not yet managing their private clouds with Bright, the article should help you see how Bright brings structure and ease-of-use to OpenStack management.

Environment

In the following article we will be configuring the hardware, CMDaemon and the Mellanox Driver.

Our example environment consists of a single head-node and 3 compute nodes in the following configuration:

- **1 head-node (controller):**
  This is where neutron-server and opensm will be running.
- **1 network-node (node001)**:
  This is where neutron agents will be running (i.e dhcp, metadata, l3 and linuxbridge).
- **2 compute hosts (node002..node003)**:
  openstack-nova-compute and libvirtd, mlnx-agent and eswitchd

Each of these servers has a ConnectX-3 InfiniBand card.

Bright Cluster Manager 7.0 has been deployed and OpenStack has been setup. Nodes are running RHEL 6.5 / CentOS 6.5

In our case deploying OpenStack resulted in the following openstack-specific configuration:

- The node category **openstack-network-nodes**. This contained node001
- The node category **openstack-compute-hosts**. This contained node002, node003
- The software image **openstack-image**. This was used by all compute nodes and also by the network node (node001..node003)

Some OpenStack deployments can have the network node configured as part of the "openstack-compute-hosts" category (and instead have an additional OpenStackNetwork role attached to the node itself). Such deployments can also be used while following the instructions in this article.

Configuration overview

The remainder of this article carries out the following steps:

- Enabling SR-IOV capabilities on the InfiniBand cards
- Configuring node categories and a software image for the hosts in cmdaemon/cmsh
- Configuring the software images (for the compute hosts, network node and virtual nodes)
- Installing the Mellanox ML2 Driver and configure its services
- Creating the InfiniBand-backed network in OpenStack Neutron
- Booting Bright-managed instances with access to the created IB network

Let's get started.

Creating The Mellanox-specific OpenStack-enabled Software Image

In this section we create a new software image. This will contain the changes needed for the IB-enabled OpenStack nodes.

On the head-node, using cmsh, we clone the existing software-image used for OpenStack nodes. In a default OpenStack deployment the software image is called “openstack-image”:

```
[headnode]% softwareimage
```

[headnode->softwareimage]% clone openstack-image openstack-image-mellanox;commit;

This creates a copy of the original software-image into the directory /cm/images/openstack-image-mellanox.

Once the clone is ready we can set the kernel parameters for this new image. Note the bolded keys/values -- these are the ones we are appending to the entire kernel parameter string:

[headnode->softwareimage[openstack-image-mellanox]]% set kernelparameters "biosdevname=0 nonm acpi=on nicdelay=0 nomodeset rdblacklist=nouveau xdriver=vesa **intel_iommu=on iommu=pt pci=realloc pci=nocrs**";commit;

The intel_iommu allows for VT-d (Intel Virtualization Technology for Directed I/O). This will allow SR-IOV devices to attach to VM instances.

OpenStack: OpenStack Neutron Mellanox ML2 Driver Configuration in Bright

Creating The Mellanox-specific OpenStack compute-hosts Category

Our compute host nodes, node002 and node003, are in the "openstack-compute-hosts" category, which is the default. In this step we clone this category and then slightly modify the clone so that it has changes needed only by compute hosts that have IB devices. We then later move the nodes that have IB, that is node002 and node003, to the new category.

Cloning the existing category for OpenStack compute hosts to the mellanox category, and then going into the Mellanox category:

[headnode]% category

[headnode->category]% clone openstack-compute-hosts openstack-compute-hosts-mellanox; use openstack-compute-hosts-mellanox;

Now we set the software image to be the software image that we have previously cloned (“openstack-image-mellanox”).

[headnode->category[openstack-compute-hosts-mellanox]]% set softwareimage openstack-image-mellanox;commit;

Then we move our two compute hosts, node002 and node003 to the fresh Mellanox-specific compute-host category:

```
[headnode->device]% device foreach -n node002..node003 (set category
openstack-compute-hosts-mellanox;commit;)
```

If not all of your compute nodes have IB devices, then you simply leave them back in their original "openstack-compute-hosts" category.

Ensure the Openstack/Mellanox Software Image is Assigned to the Network Node

In our case the network node, node001, is in the "openstack-network-nodes" category. It should therefore be enough to simply change the software image of that category to the "openstack-image-mellanox" we have created earlier:

```
headnode->category]% use openstack-network-nodes
[headnode->category[openstack-network-nodes]]% set softwareimage
openstack-image-mellanox;commit;
```

However, on some OpenStack deployments the network node might also be part of the "openstack-compute-hosts" category and only have the OpenStackNetwork role assigned on

the node level (rather than category level). In such case the network node has to have the mellanox-enabled software image assigned to it by simply setting its category to "openstack-compute-hosts-mellanox".

Enabling SR-IOV on ConnectX-3 Mellanox Infiniband Cards

In this step we will update the firmware on IB devices and enable SR-IOV. It is recommended to use the latest available OFED and firmware. At the time of writing that was:

- MLNX_OFED_LINUX-2.3-1.0.1-rhel6.5-x86_64.tgz OFED, and
- IB devices firmware version 2.32.5100.

You can choose to skip this section if the firmware and the OFED package are already updated.

Download the latest Mellanox OFED package for Centos/RHEL 6.5

http://www.mellanox.com/page/products_dyn?product_family=26&mtag=linux_sw_drivers

The package name looks like this: MLNX_OFED_LINUX-<version>-rhel6.5-x86_64 (The package can be download either as an ISO or a tarball).

The OFED package is to be copied (one way or another) to all the compute hosts which require an upgrade of the firmware. *(Note, only during a later stage of the article we will be describing the actual installation of the OFED in the package into the software images. Right now we only want the file on the live node)*

An efficient way to upgrade the firmware on multiple hosts would be to extract (in case of tar.gz file) or copy (in case of using a ISO) the OFED package directory to a shared location such as /cm/shared (which is mounted on compute nodes by default).
Then we can use the pdsh tool in combination with category names to parallelize the upgrade.

In our example we extract the OFED package to /cm/shared/ofed.

Before we begin the upgrade we need to remove the cm-config-intelcompliance-slave package to avoid conflicts:

```
[root@headnode ~]# pdsh -g category=openstack-compute-hosts-mellanox "yum remove -y cm-config-intelcompliance-slave"
```

(For now we will only remove it from live nodes. We will remove it from the software image later in the article. Do not forget to also run this command on the headnode)

In some cases the package 'qlgc-ofed.x86_64' may also need to be removed. In such case the

mlnxofed install will not proceed. A log of the installer can always be viewed in /tmp/MLNX_OFED_LINUX-<version>.<num>.logs/ofed_uninstall.log to determine which package is conflicting and remove it manually.

And then run the firmware upgrade:

```
[root@headnode ~]# pdsh -g category=openstack-compute-hosts-mellanox
"cd /cm/shared/ofed/MLNX_OFED_LINUX-2.3-1.0.1-rhel6.5-x86_64/ && echo
\"y\" | ./mlnxofedinstall --enable-sriov" | tee -a
/tmp/mlnx-firmware-upgrade.log
```

(*Do not forget to execute these two steps on the network node and the headnode*)

Note that we are outputting both to the screen and to a temporary file (/tmp/mlnx-firmware-upgrade.log). This can help spotting any errors that might occur during the upgrade.

Running the 'mlnxofedinstall --enable-sriov' utility does two things:

- installs OFED on the live nodes
- updates the firmware on the InfiniBand cards and enables the SR-IOV functionality.

Notice, that in the case of compute nodes (node001-node003) at this point we're mostly interested in the latter (firmware update and enabling SR-IOV). Since we've run this command on the live node, the filesystem changes have not been propagated to the software image used by the nodes (i.e. at this point they would be lost on reboot). We will take care of that later on in this article by installing the OFED also to the software image.

In the case of headnode, however, running this command also effectively installs OFED and update firmware, which is exactly what we want.

Configuring the Software Images

The Virtual-Nodes-Image

The simplest would be to begin with the software image used for our virtual nodes (Bright-managed nodes), in our example: “virtual-node-image”. We must install the MLNX OFED into it.

We extract and install the MLNX OFED package downloaded earlier into the virtual nodes software image.

For example:

```
# cd /cm/images/virtual-node-image/opt/

# tar zxvf /opt/MLNX_OFED_LINUX-<version>-rhel6.5-x86_64.tgz

# chroot /cm/images/virtual-node-image/

# yum remove cm-config-intelcompliance-slave -y

# cd opt/MLNX_OFED_LINUX-<version>-rhel6.5-x86_64/

# ./mlnxofedinstall

# exit
```

Note that running ./mlnxofedinstall will report errors due to inability to detect the IB card inside the chroot of the software image. The errors can be ignored (i.e. "Failed to update Firmware")

We are done setting up the virtual node software image.

The OpenStack-Image-Mellanox

MLNX OFED has to be installed onto the Mellanox software image, for example:

```
# cd /cm/images/openstack-image-mellanox/opt/

# tar zxvf /opt/MLNX_OFED_LINUX-<version>-rhel6.5-x86_64.tgz

# chroot /cm/images/openstack-image-mellanox/

# yum remove cm-config-intelcompliance-slave -y

# cd /opt/MLNX_OFED_LINUX-<version>-rhel6.5-x86_64/

# ./mlnxofedinstall
```

Again, ignore errors reported by running ./mlnxofedinstalls inside chroot.

At this point we can already add the Neutron Mellanox ML2 Driver software into the "openstack-image-mellanox" software image (while we are still chrooted in the software-image's directory):

Install the Mellanox IceHouse repository:

wget -O /etc/yum.repos.d/mlnx-icehouse.repo http://www.mellanox.com/downloads/solutions/openstack/icehouse/repo/mlnx-icehouse/mlnx-icehouse.repo

Install the eswitchd RPM:

yum install eswitchd

Install the required RPM for the Neutron agent:

yum install openstack-neutron-mellanox

Run the following command:

touch /var/log/eswitchd.log && chown eswitch.eswitch /var/log/eswitchd.log

And:

```
# yum install mlnxvif -y
```

Install the Mellanox dnsmasq driver:

yum install -y http://www.mellanox.com/downloads/solutions/openstack/icehouse/repo/mlnx-icehouse/mlnx-dnsmasq-2014.1.1-1.noarch.rpm

Install (upgrade) the dnsmasq package to 2.66-3.1:

yum install -y ftp://ftp.pbone.net/mirror/ftp5.gwdg.de/pub/opensuse/repositories/home:/kalyaka/CentOS_CentOS-6/x86_64/dnsmasq-2.66-3.1.x86_64.rpm

At this point you can exit the chroot.

There is currently a chance that neutron-mlnx-agent will fail to start-up because it cannot find eswitchd. It takes eswitchd a moment before its daemon comes up, in which time the neutron-mlnx-agent cannot find it.

We resolve this with a temporary fix. Add the "sleep 3" line to the "start" function in the file /cm/images/openstack-image-mellanox/etc/init.d/neutron-mlnx-agent :

```
start() {
    [ -x $exec ] || exit 5
    for config in ${configs[@]}; do
        [ -f $config ] || exit 6
    done
    sleep 3
    echo -n $"Starting $prog: "
    daemon --user neutron --pidfile $pidfile "$exec --log-file /var/log/$proj/$plugin.log ${configs[@]/#/--config-file } &>/dev/null & echo \$! >
$pidfile"
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}
```

Add/edit the file /cm/images/openstack-image-mellanox/etc/modprobe.d/mlx4_core.conf :

options mlx4_core port_type_array=1,1 num_vfs=16 probe_vf=0 debug_level=1

Add/edit the file /cm/images/openstack-image-mellanox/etc/modprobe.d/mlx4_ib.conf :

options mlx4_ib sm_guid_assign=0

Add/edit the file /cm/images/openstack-image-mellanox/etc/infiniband/openib.conf :

```
# Start HCA driver upon boot
```

```
ONBOOT=yes

# Node description

NODE_DESC=$(hostname -s)

# Wait for NODE_DESC_TIME_BEFORE_UPDATE sec before node_desc update

NODE_DESC_TIME_BEFORE_UPDATE=20

# Max time in seconds to wait for node's hostname to be set

NODE_DESC_UPDATE_TIMEOUT=120

# Seconds to sleep after openibd start finished and before releasing the shell

POST_START_DELAY=0

# Set rx_channels/tx_channels to 1 to disable IPoIB RSS/TSS

SET_IPOIB_CHANNELS=no

# Run /usr/sbin/mlnx_affinity

RUN_AFFINITY_TUNER=no

# Run /usr/sbin/mlnx_tune

RUN_MLNX_TUNE=no

# Load UMAD module

UMAD_LOAD=yes

# Load UVERBS module

UVERBS_LOAD=yes

# Load UCM module

UCM_LOAD=yes

# Load RDMA_CM module

RDMA_CM_LOAD=yes

# Load RDMA_UCM module
```

```
RDMA_UCM_LOAD=yes

# Increase ib_mad thread priority

RENICE_IB_MAD=no

# Run sysctl performance tuning script

RUN_SYSCTL=no

# Load MTHCA

MTHCA_LOAD=no

# Load MLX4 modules

MLX4_LOAD=yes

# Load MLX5 modules

MLX5_LOAD=yes

# Load MLX4_EN module

MLX4_EN_LOAD=yes

# Load MLX4_VNIC module

MLX4_VNIC_LOAD=no

# Load IPoIB

IPOIB_LOAD=yes

# Enable IPoIB Connected Mode

SET_IPOIB_CM=auto

# Load E_IPoIB

E_IPOIB_LOAD=yes

# Load SDP module

SDP_LOAD=no

# Load SRP module
```

```
SRP_LOAD=no

# Load RDS module

RDS_LOAD=no

# Load QIB

QIB_LOAD=no

# Load IPATH

IPATH_LOAD=no

# Load QLogic VNIC module

QLGC_VNIC_LOAD=no

# Load CXGB3 modules

CXGB3_LOAD=no

# Load CXGB4 modules

CXGB4_LOAD=no

# Load NES modules

NES_LOAD=no

# Enable SRP High Availability daemon

SRPHA_ENABLE=no

SRP_DAEMON_ENABLE=no
```

Reboot the Nodes

At this point we can reboot the nodes node001..node003 (make sure that the "nextinstallmethod" for the nodes is set to FULL to ensure that the nodes get synchronised with the new software image).

[headnode]% device foreach -n node001..node003 (reboot)

(If these nodes are used as Ceph OSD's, and depending on how many OSD's are used in the cluster, make sure to reboot them one after another to prevent any data loss in your Ceph cluster.)

Once the nodes are up you can verify whether the SR-IOV is enabled. You should notice 16 Virtual Function ConnectX-3 devices:

[root@node001 ~]# lspci |grep -i mel

05:00.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3]

05:00.1 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.2 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.3 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.4 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.5 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.6 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:00.7 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.1 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.2 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.3 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.4 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.5 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.6 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:01.7 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

05:02.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]

Depending on the number of IB ports, you should also notice a number of newly created eth devices. (You can check how many IB ports exist on the Mellanox interface using the command ‘ibstat’ on the compute host).

The eth devices get assigned the highest number available. In our case we find eth6 and eth7 have been created. (Take note of those interface names, you will need them later). Running the ethtool on one of the devices produces:

ethtool -i eth6

driver: eth_ipoib

version: 2.3-1.0.0 (Sep 8 2014)

firmware-version: 1

bus-info: ib0

supports-statistics: yes

supports-test: no

supports-eeprom-access: no

supports-register-dump: no

supports-priv-flags: no

Check Your Config

To ensure the BIOS has been set with IOMMU (VT-d) you should see output running the following command on the compute hosts:

```
# dmesg | grep -e DMAR -e IOMMU
```

To ensure that SR-IOV is enabled one can run:

```
# mst start

# flint -d /dev/mst/mt4099_pciconf0 dc
```

the [HCA] section of the output should show (note the sriov_en = true):

```
[HCA]

hca_header_device_id = 0x1003

hca_header_subsystem_id = 0x0006

dpdp_en = true

eth_xfi_en = true

mdio_en_port1 = 0

num_pfs = 1

total_vfs = 16 // Total number of VMs

sriov_en = true // SR-IOV is enabled
```

Configuring opensm on the Controller (headnode)

OpenStack: OpenStack Neutron Mellanox ML2 Driver Configuration in Bright

In a typical Bright deployment with InfiniBand, you might find that the opensm is one of services assigned to the network node. In our case we move opensm service to run on the head-node.

This can be done in the services configuration of the head-node in cmsh located in:

[headnode]% device use master;services

```
[headnode->device[headnode]->services]% status opensm
opensm          [   UP   ]
```

And we would like to stop it if it is running on the network node:

```
[headnode->device[headnode]->services]% device use node001
[headnode->device[node001]->services]% status opensm
opensm          [   UP   ]
[headnode->device[node001]->services]% stop opensm
Fri Oct 17 08:42:09 2014 [notice] node001: Service opensm was stopped
```

(Note that you might find both /etc/init.d/opensm and /etc/init.d/opensmd but they both point to the same daemon).

Note that internally the Mellanox ML2 Mechanism Driver makes use of the facilities provided by the "vlan" ML2 network type driver. This effectively means that the concept of VLANs and VLAN IDs is used to represent the IB segments and IB Segment IDs. Hence the references to VLANs in the following paragraphs.

There are two files that have to be added/edited before starting (or restarting) the opensm daemon:

The /etc/opensm/partitions.conf file defines mappings of vlan (segmentation IDs) to InfiniBand partitions.

For example: If our vlan range for the IB network will be 1 to 10 we configure the file as follows: (the range will be configured later on in /etc/neutron/plugins/mlnx/mlnx_conf.ini, i.e. network_vlan_ranges = ibhostnet:1:10)

/etc/opensm/partitions.conf :

management=0x7fff,ipoib, sl=0, defmember=full : ALL, ALL_SWITCHES=full,SELF=full;

vlan1=0x1, ipoib, sl=0, defmember=full : ALL_CAS;

vlan2=0x2, ipoib, sl=0, defmember=full : ALL_CAS;

vlan3=0x3, ipoib, sl=0, defmember=full : ALL_CAS;

vlan4=0x4, ipoib, sl=0, defmember=full : ALL_CAS;

vlan5=0x5, ipoib, sl=0, defmember=full : ALL_CAS;

vlan6=0x6, ipoib, sl=0, defmember=full : ALL_CAS;

vlan7=0x7, ipoib, sl=0, defmember=full : ALL_CAS;

vlan8=0x8, ipoib, sl=0, defmember=full : ALL_CAS;

vlan9=0x9, ipoib, sl=0, defmember=full : ALL_CAS;

vlan10=0xa, ipoib, sl=0, defmember=full : ALL_CAS;

And we add/edit /etc/opensm/opensm.conf to include:

allow_both_pkeys TRUE

Now the opensm service can be restarted. This can be done from the head-node's services:

[headnode]% device use master;services

[headnode->device[masterdoop]->services]% restart opensm

Configure the Mellanox Neutron ML2 Driver

We need to override some of OpenStack's configuration files, where some of the keys/values are controlled by CMDaemon.

We can do this in one of two ways: overwriting specific keys/values via the OpenStackNode role in cmsh (only CMDaemon revision r22282 or higher) or by freezing the file using the FreezeFile value in /cm/local/apps/cmd/etc/cmd.conf, restarting the CMDaemon, and then editing the files manually. The downside of the latter is that CMDaemon will not be able to update the files in case something has to be updated or appended (new key/values for example). Whereas the former will not work with older CMDaemon and ini section names containing underscores.

The first file to edit is located on the head-node: /etc/neutron/plugins/ml2/ml2_conf.ini .

This is an example of how the file should look like:

```
[ml2]
type_drivers=flat,vlan
tenant_network_types=flat,vlan
mechanism_drivers=mlnx,linuxbridge,l2population
[ml2_type_flat]
flat_networks=phyflatnet
[ml2_type_vlan]
network_vlan_ranges=ibhostnet:1:10,phyvlanhostnet:20:30
[ml2_type_gre]
[ml2_type_vxlan]
[securitygroup]
enable_security_group = True
firewall_driver=dummy_value_do_not_remove
[eswitch]
vnic_type=hostdev
apply_profile_patch=True
```

Note that in the above configuration we are using two type_drivers: flat and vlan, and have configured those accordingly in their respective sections. Most importantly we are using the "mlnx" and "linuxbridge" as the mechanism drivers.

For the vlan we specify two physical vlan networks: The ‘ibhostnet:1:10’ defines the IB network which will ultimately be used by the Mellanox Neutron driver (Instances booted onto this network will have a Virtual Function (SR-IOV) HCA attached to them) and the ‘phyvlanhostnet:20:30’ is an ethernet based vlan network. Both VLAN ranges used are only examples and you should change them to reflect your own network config.

Instead of freezing the file, we can use the role. Using the role, we can overwrite the keys/values as follows:

[headnode]% device use master;roles;use openstacknode;

[headnode->device[masterdoop]->roles[openstacknode]]% set configfilemodifications "-/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]type_drivers" "+/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]type_drivers=flat,vlan" "-/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]tenant_network_types" "+/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]tenant_network_types=flat,vlan" "-/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]mechanism_drivers" "+/etc/neutron/plugins/ml2/ml2_conf.ini[ml2]mechanism_drivers=mlnx,linuxbridge,l2population" "-/etc/neutron/plugins/ml2/ml2_conf.ini[ml2_type_vlan]network_vlan_ranges" "+/etc/neutron/plugins/ml2/ml2_conf.ini[ml2_type_vlan]network_vlan_ranges=ibhostnet:1:10,phyvlanhostnet:11:20" "+/etc/neutron/plugins/ml2/ml2_conf.ini[eswitch]vnic_type=hostdev" "+/etc/neutron/plugins/ml2/ml2_conf.ini[eswitch]apply_profile_patch=True";commit;

Here's how the /etc/neutron/plugins/ml2/ml2_conf.ini file should look like when used for deployments with VXLAN-based network isolation. (ignore the 'vxlan' if you're using VLAN-based network isolation only)

[ml2]

type_drivers=flat,vlan,vxlan

tenant_network_types=flat,vlan,vxlan

mechanism_drivers=mlnx,linuxbridge,l2population

[ml2_type_flat]

flat_networks=phyflatnet

[ml2_type_vlan]

network_vlan_ranges=phyvlanhostnet:11:20

[ml2_type_gre]

[ml2_type_vxlan]

vni_ranges=1:8000

vxlan_group=224.0.0.1

[securitygroup]

enable_security_group = True

firewall_driver=dummy_value_do_not_remove

[eswitch]

vnic_type=hostdev

apply_profile_patch=True

The next file to edit is /etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini, located on each compute host, and can therefore be edited via the openstack-compute-hosts-mellanox category roles (or alternatively freeze the file and edit it in the software-image as noted in the first file's example above).

In our example, the file should look like this:

[vlans]

tenant_network_type=flat,vlan

network_vlan_ranges=ibhostnet:1:10,phyvlanhostnet:11:20

[linux_bridge]

physical_interface_mappings=ibhostnet:eth6,phyflatnet:br0-flat-veth,phyvlanhostnet:br0-veth

[vxlan]

enable_vxlan=False

[agent]

[securitygroup]

firewall_driver=neutron.agent.linux.iptables_firewall.IptablesFirewallDriver

If you're using VXLANs for network isolation instead of VLANs, you will want to have the following [vxlan] section:

```
enable_vxlan=True

local_ip=10.161.0.2         # IP of this node on the underlying VXLAN
network (can be physical or alias interface)

vxlan_group=224.0.0.1       # VXLAN multicast IP, should be the same
for all nodes

l2_population=True
```

For VXLAN isolation you will also want to remove the "phyvlanhostnet:x" entry from the "physical_interface_mappings" key and remove the "vlan" entry from the "tenant_network_type" key. (note, adding "vxlan" into it instead is NOT required)

Note that we are defining here the two physical vlan networks we have defined in the previous file (ml2_conf.ini).

This file can also be configured via the category’s role:

[headnode]% category use openstack-compute-hosts-mellanox;roles;use openstacknode;

[headnode>category[openstack-compute-hosts-mellanox]->roles[openstacknode]]% set configfilemodifications "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]tenant_network_type" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]tenant_network_type=flat,vlan" "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]network_vlan_ranges" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]network_vlan_ranges=ibhostnet:1:10,phyvlanho

stnet:11:20" "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[linux_bridge]physical_interface_mappings" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[linux_bridge]physical_interface_mappings=ibhostnet:**eth6**,phyflatnet:br0-flat-veth,phyvlanhostnet:eth0";commit;

What is very important to note here is the *physical_interface_mappings* setting: the 'ibhostnet' is mapped to eth6. This is the interface we detected after rebooting the nodes. This interface's number can differ on each host depending on the number assigned to the eth created by Mellanox. If a node is using a different eth (e.g. eth4) you should assign the 'configfilemodifications' on the node's role and not via the category. This will override the category's 'configfilemodifications' setting and you will be able to specify the correct eth interface. In our example, node003 has the eth4 created by Mellanox. We set the *physical_interface_mappings* accordingly:

[headnode]% device use node003

[headnode->device[node003]]% roles

[headnode->device[node003]->roles]% assign openstacknode

[headnode->device*[node003*]->roles*[openstacknode*]]% set configfilemodifications "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]tenant_network_type" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]tenant_network_type=flat,vlan" "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]network_vlan_ranges" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[vlans]network_vlan_ranges=ibhostnet:1:10,phyvlanhostnet:11:20" "-/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[linux_bridge]physical_interface_mappings" "+/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini[linux_bridge]physical_interface_mappings=ibhostnet:**eth4**,phyflatnet:br0-flat-veth,phyvlanhostnet:eth0";commit;

The /etc/neutron/dhcp_agent.ini file has to be edited. We can simply do this on the software image (/cm/images/openstack-image-mellanox/etc/neutron/dhcp_agent.ini):

[DEFAULT]

dhcp_driver = mlnx_dhcp.MlnxDnsmasq

enable_isolated_metadata=True

dnsmasq_config_file=/etc/neutron/dnsmasq.bright.conf

interface_driver=neutron.agent.linux.interface.BridgeInterfaceDriver

The /etc/nova/nova.conf on the software image has to be edited (/cm/images/openstack-image-mellanox/etc/nova/nova.conf). We add the vif_driver definition in the [libvirt] section:

...
[libvirt]

vif_driver=mlnxvif.vif.MlxEthVIFDriver

...

We edit the mlnx-agent file on the software image /cm/images/openstack-image-mellanox/etc/neutron/plugins/mlnx/mlnx_conf.ini :

[mlnx]

verbose = True

debug = True

tenant_network_type = vlan

network_vlan_ranges = ibhostnet:1:10

physical_network_type_mappings = ibhostnet:ib

physical_network_type = ib

[eswitch]

physical_interface_mappings = ibhostnet:ib0

vnic_type = hostdev

daemon_endpoint = 'tcp://localhost:60001'

[agent]

polling_interval = 2

rpc_support_old_agents = True

[securitygroup]

enable_security_group = True

And eswitchd in the software image /cm/images/openstack-image-mellanox/etc/eswitchd/eswitchd.conf file:

```
[DEFAULT]

default_log_levels="eswitchd=DEBUG"

verbose=True

debug=True

log_file=/var/log/eswitchd.log

[DAEMON]

fabrics = 'ibhostnet:ib0'

default_timeout=4000

socket_os_port=60001

socket_of_port=60000
```

In the category ‘openstack-compute-hosts-mellanox’ we add the eswitchd and neutron-mlxn-agent services to autostart. For example:

```
[headnode]% category use openstack-compute-hosts-mellanox;services

[headnode->category[openstack-compute-hosts-mellanox]->services]% add eswitchd

[headnode->category*[openstack-compute-hosts-mellanox*]->services*[eswitchd*]]% set autostart yes

[headnode->category*[openstack-compute-hosts-mellanox*]->services*[eswitchd*]]% set monitored yes

[headnode->category*[openstack-compute-hosts-mellanox*]->services*[eswitchd*]]% commit
```

Repeat the above example for the neutron-mlnx-agent.

At this point you should set the *nextinstallmethod* of the nodes to FULL to make sure all the recent changes will be synchronised during the next reboot.

Now you can proceed to reboot the network-node and the compute-hosts (same as before: be careful not to reboot all the nodes at the same time if they are also used as ceph OSDs).

Verifying the Configuration

After having rebooted the nodes we can run the following command to ensure that the neutron-mlnx-agent is running on all of the Mellanox compute nodes:

```
# neutron agent-list
+--------------------------------------+---------------------+---------+-------+----------------+
| id                                   | agent_type          | host    | alive | admin_state_up |
+--------------------------------------+---------------------+---------+-------+----------------+
| 4d48dd6e-1cc0-456e-b3b8-6d0f967ff330 | Linux bridge agent  | node003 | :-)   | True           |
| 7a32c1fa-afa5-47f0-8d52-4673b7f2152e | DHCP agent          | node001 | :-)   | True           |
| 84077a6b-4691-4247-b9a3-270cb2fc3b26 | L3 agent            | node001 | :-)   | True           |
| 91c809ba-5c0b-44af-bddf-3214fc42f1d2 | Mellanox plugin agent | node002 | :-)   | True           |
| 93bd20f0-3dad-469e-88da-6211190e43ec | Metadata agent      | node001 | :-)   | True           |
| a70d1852-0f96-4421-bf7d-d1b2559d5768 | Linux bridge agent  | node001 | :-)   | True           |
| b9f57436-63ab-4f9f-95cb-e981973bd4c5 | Mellanox plugin agent | node001 | :-)   | True           |
| c81c9b74-0584-4390-95f3-d547b08d4d89 | Linux bridge agent  | node002 | :-)   | True           |
| fb93caa9-28af-4e49-bcf2-7f2ee9a7ea84 | Mellanox plugin agent | node003 | :-)   | True           |
+--------------------------------------+---------------------+---------+-------+----------------+
```

If you find that the mlnx-agent is marked with ‘xxx’ you can ssh into that node and check the /var/log/neutron/mlnx-agent.log and/or the /var/log/eswitchd.log to find out why the agent is not functioning.

If you find that the linux-bridge agent is marked with ‘xxx’, you should check the /var/log/neutron/linuxbridge-agent.log . A common reason for it to fail is having a non-existent eth device defined in the /etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini configuration file.

Configuring the Default Provider Network

When a user creates a network object in OpenStack, that network is backed by a specific provider network. In the case of multiple provider networks (e.g. IB network + VLAN segment, or IB Network + VXLAN segment), the dashboard in OpenStack does not let users pick which provider network backend is to be used. If users are expected to use OpenStack via the dashboard it is therefore important to configure OpenStack in a way which will make it default to the provider network type which is expected/preferable by the users.

In the case of IB Networks and VXLAN networking, the default provider network can be influenced by changing the order of network types in the /etc/neutron/plugin.ini file on the head-node. In the case of

```
tenant_network_types=flat,vxlan,vlan
```

The default network type picked by the dashboard will be VXLAN. but with

```
tenant_network_types=flat,vlan,vxlan
```

The default will be VLAN, which in our case represents the IB network.

This is trickier in the case of IBNetworks and VLAN based networks, since in that case we are talking about two provider networks of the network type VLAN (and no VXLAN networks). In such case using tenant_network_type is not enough, and what's important is the alphanumeric ordering of the name of the name of the VLAN networks. In the case of two provider network called "ibhostnet" and "phyvlanhostnet", the "ibhostnet" comes first in alphanumeric order, which means the networks created by users will by default be backed with the "ibhostnet" provider network. Should one want to the "phyvlanhostnet" to be the default, the "phyvlanhostnet" should instead be renamed to something which will put it before "ibhostnet" in the alphanumeric ordering, e.g. to "aphyvlanhostnet" or "01_phyvlanhostnet"

Note: Although not possible via the Dashboard, it is possible to pick the provider network when

creating a openstack network using the Neutron API or the command line tool (neutron net-create). But in order for the regular (non-admin) user to be able to specify that the /etc/neutron/policy.json must be modified appropriately (and then neutron-server restarted).

Creating OpenStack Network Backed by IB

It is now possible to create an OpenStack network object which will be used by our Bright-managed instances in the following step. Since in our example we are configuring IB networks (VLAN) + VLAN isolation, and since we cannot pick the provider network type from the dashboard (as explained in the previous section), we must use the command line tool to create a virtual network backed by IB.

To create a network backed InfiniBand provider network:

neutron net-create --provider:network_type=vlan --provider:physical_network=ibhostnet --provider:segmentation_id=1 --shared --router:external=false ibnetwork

Following the creation of the network on the command-line it is also necessary to create a subnet. It is up to you whether you would like to create subnets using the command-line or the OpenStack dashboard.

Note that a network should be set to ‘shared’ (as it is in example above) if it is going to be used by more than just the tenant who originally created it.

[EXAMPLE] This is just as an example. To create a network backed with Ethernet one could use the following command:

 # neutron net-create --provider:network_type=vlan --provider:physical_network=phyvlanhostnet --provider:segmentation_id=11 --router:external=false vlantestnet001

To add a subnet to the ibnetwork from the command-line we can run something like this:

```
[root@headnode ~]# neutron subnet-create ibnetwork 192.168.80.0/24
--name ibsubnet
```

Configuring Bright-managed instances with InfiniBand Interfaces

In order for us to be able to boot Bright-managed instances we must create a bright network object representing the openstack network object 'ibnetwork' we have created in the previous section.

We create a new network in cmsh and make sure its name corresponds to the name of the network we have created in neutron for the ibhostnet (in the example from the previous section we create the network called ‘ibnetwork’).

Other values such as IP ranges and gateway should correspond to the subnet you have created in Neutron (or the OpenStack dashboard) for this network.

[headnode->network]% add ibnetwork

[headnode->network*[ibnetwork*]]% set openstacknetworktype other

[headnode->network*[ibnetwork*]]% set openstackvlanrange 1:10

[headnode->network*[ibnetwork*]]% set openstacknetworkisshared 1

[headnode->network*[ibnetwork*]]% set openstackphysicalnetworkname ibnetwork

[headnode->network*[ibnetwork*]]% set domainname ibhostnet.cluster

[headnode->network*[ibnetwork*]]% set baseaddress 192.168.80.0

[headnode->network*[ibnetwork*]]% set netmaskbits 24

[headnode->network*[ibnetwork*]]% set broadcastaddress 192.168.80.255

[headnode->network*[ibnetwork*]]% commit

Which will result in the following configuration:

```
[headnode->network[ibnetwork]]% show
Parameter
Value
--------------------------------
------------------------------------------------
OpenStack Host
network
OpenStack Physical network name
```

```
ibnetwork
OpenStack VLAN ID
0
OpenStack VLAN Range
1:10
OpenStack alloc pool end
0.0.0.0
OpenStack alloc pool start
0.0.0.0
OpenStack network

OpenStack network is shared
yes
OpenStack network type
Other
Revision

Base address
192.168.80.0
Broadcast address
192.168.80.255
Domain Name
ibhostnet.cluster
Dynamic range start
0.0.0.0
Dynamic range end
0.0.0.0
Gateway
0.0.0.0
IPv6
no
Lock down dhcpd
no
MTU
1500
Management allowed
no
Netmask bits
24
Node booting
no
Notes                                <0
bytes>
Type
Internal
name
ibnetwork
EC2AvailabilityZone
```

```
EC2SubnetID

Private Cloud
```

Once the network is defined, we can add an interface to the vnodes to make use of the IB network (vnodes refer to virtual nodes defined in cmdaemon/cmsh). If you have deployed OpenStack with Bright-managed instances enabled, you will already have several "vnode" nodes configured in your Bright cluster's configuration. We reconfigure them to ensure they get the ib interfaces when they get instantiated inside OpenStack:

device foreach -n vnode001..vnode012 (interfaces;add physical ib0;set dhcp yes;set network ibnetwork ;commit;)

Since any changes made to vnode's configuration only take place when the vnode is first instantiated in OpenStack, if your vnodes have already been instantiated inside openstack (if they have an InstanceID set), you will have to terminate them, and only then power them on.

[headnode->device[vnode001]]% virtualnodesettings

[headnode->device[vnode001]->virtualnodesettings]% get instanceid

<should return empty line>

If all goes well, on the compute host where the vm is being launched we should see such lines in the /var/log/neutron/mlnx-agent.log:

```
2014-10-17 09:26:11.265 8745 INFO
neutron.plugins.mlnx.agent.eswitch_neutron_agent [-] Port
fa:16:3e:ec:98:dc updated
2014-10-17 09:26:11.269 8745 INFO
neutron.plugins.mlnx.agent.eswitch_neutron_agent [-] Provisioning
network c4fb0bb2-c4e2-4863-998a-ee4ac0123f24
2014-10-17 09:26:11.269 8745 INFO
neutron.plugins.mlnx.agent.eswitch_neutron_agent [-] Binding
Segmentation ID 1to eSwitch for vNIC mac_address fa:16:3e:ec:98:dc
```

Once the vnodes booted up, you can simply SSH to them (or use the console) and verify that

the ib0 interface has been added:

```
[root@vnode001 ~]# lspci |grep -i mel

00:05.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]
```

And run:

```
[root@vnode001 ~]# ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

    inet 127.0.0.1/8 scope host lo

    inet6 ::1/128 scope host

       valid_lft forever preferred_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000

    link/ether fa:16:3e:3c:40:4a brd ff:ff:ff:ff:ff:ff

    inet 10.141.96.21/16 brd 10.141.255.255 scope global eth0

    inet6 fe80::f816:3eff:fe3c:404a/64 scope link

       valid_lft forever preferred_lft forever

3: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2044 qdisc mq state UP qlen 1024

    link/infiniband a0:00:0e:04:fe:80:00:00:00:00:00:00:00:fa:16:3e:00:00:0a:ec:e7 brd 00:ff:ff:ff:ff:12:40:1b:80:01:00:00:00:00:00:00:ff:ff:ff:ff

    inet 192.168.80.17/24 brd 192.168.80.255 scope global ib0

    inet6 fe80::fa16:3e00:a:ece7/64 scope link

       valid_lft forever preferred_lft forever

4: ib1: <BROADCAST,MULTICAST> mtu 4092 qdisc noop state DOWN qlen 1024

    link/infiniband a0:00:0e:08:fe:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 brd
```

00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff

Note the ib0 interface with the IP on the subnet we have added to the ibnetwork(ibhostnet) earlier. We will also see the ib1 interface in the output since our device is dual port, but we won't be able to use it out of the box because:

-ib1 on the compute host is not in use (down)

-ib1 will be missing ifcfg file,

-ib1 hasn't been specified in the /cm/images/openstack-image-mellanox/etc/neutron/plugins/mlnx/mlnx_conf.ini file in an earlier stage, therefore has no network connectivity to the underlying fabric.

It is also possible to boot instances which are not Bright-managed onto the IB network. In order for this to work you will have to make sure that the software image you are using has the MLNX OFED installed.

This is in order for the guest OS to boot up be able to configure the device. You will probably also have to add a /etc/sysconfig/network-scripts/ifcfg-ib0 file, for example:

DEVICE=ib0

ONBOOT=yes

MTU=65520

CONNECTED_MODE=yes

BOOTPROTO=dhcp

PREFIX=24

Wrapup

You should now be able to start several vnodes on the same network and run MPI jobs across of them.

Note, that utilities which require to send MADs to the SM will not work on VMs by design (e.g. ibhosts, ibnetdiscover and ibping). This was done in order to protect the fabric from a VM to manipulate the fabric and to run SM.

OpenStack: OpenStack Neutron Mellanox ML2 Driver Configuration in Bright

Closing notes:

When introducing changes in the config files by freezeing them, make sure to have those files recreated on the nodes after performing a FULL install on the node. Doing a FULL install will wipe the system drive, and any local changes made onto it. Thus, in it's reasonable to introduce the changes which apply to all compute nodes in the nova.conf located in the software image, and then use the finalize script of the node-installer environemnt to introduce changes which are specific to nodes. Obviously, the config file modified in this way should be marked as frozen in cmd.conf, otherwise, CMDaemon will attempt to overwrite them once the node starts.

Sources:

http://community.mellanox.com/docs/DOC-1317

https://wiki.openstack.org/wiki/Mellanox-Neutron-Icehouse-Redhat

http://support.brightcomputing.com/manuals/7.0/openstack-deployment-manual.pdf

Unique solution ID: #1238
Author: Nuriel
Last update: 2015-02-14 00:04