MIRANTIS

Network FunctionsVirtualization Solution Guide for DevStack

version 9.2

Contents

Copyright notice

Preface

This documentation provides information on how to use Fuel to deploy OpenStack environments. The information is for reference purposes and is subject to change.

Intended Audience

This documentation is intended for OpenStack administrators and developers; it assumes that you have experience with network and cloud concepts.

Documentation History

The following table lists the released revisions of this documentation:

Revision Date	Description
February 6, 2017	9.2 GA

Introduction

This guide describes procedures required to enable Network Functions Virtualization (NFV) in the DevStack version of OpenStack. With enabled NFV features, DevStack can act as NFV infrastructure (NFVI) enabling Virtual Network Functions (VNFs).

The objective of NFV is to address problems related to communications service providers (CSPs) that need to deliver specialized network functions such as firewalls, load balancers, and content filters on an on-demand basis. Rather than requiring the deployment of proprietary hardware appliances in their network, NFV helps CSPs to address these problems by decoupling software functions from the underlying hardware. NFV performs this decoupling by leveraging standard IT virtualization technology along with the introduction of open APIs and an ecosystem of suppliers, building end-to-end, flexible, and scalable solutions. In such way, NFV improves hardware resources utilization and reduces operating costs.

NFV features can be added to Compute and Networking services to build NFVI on top of the OpenStack environment. Fusion of NFV and OpenStack helps to deliver the virtualized network function in a quicker and automated way.

The guide describes the following NFV features that can deliver near bare-metal performance for the virtual function in the OpenStack infrastructure:

- Huge Pages is a technology that supports of 2MB and 1GB size memory pages. Huge Pages reduce time to access data stored in the memory by using bigger memory pages, which leads to fewer page entries to look up by CPU when choosing a page associated with a current process.
- NUMA/CPU pinning is a shared memory architecture that describes the placement of main memory modules on processors in a multiprocessor system. You can leverage NUMA when you have data strongly associated with certain tasks or users. In such case, CPU can use its local memory module to access data reducing access time.
- SR/IOV is an extension to the PCI Express (PCIe) specification that enables a network adapter to separate access to its resources among various PCIe hardware functions: Physical Function (PF) and Virtual Functions (VFs). As a result, you can achieve near bare-metal performance, since network devices can forward traffic directly to a VF bypassing the host.
- DPDK is a set of libraries and drivers to perform fast packet processing in the user space that OVS can use to move network packets processing from a kernel to a user space. OVS with DPDK acceleration on compute nodes reduces the processing time of network packets transferred between a host's network interface and a guest bypassing the host's kernel.

Prerequisites

This section describes software and hardware requirements needed to enable NFV features as well as possible limitations.

Software requirements

Software requirements

NFV feature	Linux distribution	Linux kernel version	libvirt version	QEMU version	OpenStack version
DPDK	Ubuntu 14.04	2.6.33 or later	1.2.9.3 or later	2.3.0	Liberty
Huge Pages	Ubuntu 14.04	default	1.2.8 or later	2.1.0 or later	Liberty
NUMA	Ubuntu 14.04	default	1.2.7 or later (except 1.2.9.2)	2.1.0 or later	Liberty
SR-IOV	Ubuntu 14.04, Fedora 22 or newer, or CentOS/RHEL 7	3.8 or later	N/A	N/A	Liberty

Use the latest DPDK version 16.04 or the master branch to avoid memory segmentation fault that DPDK-enabled applications may cause in a VM instance on the host where Open vSwitch with DPDK is enabled.

DPDK has the following extra requirements for kernel:

- UIO or vfio support
- HUGETLBFS support

Seealso

LP1597004: DPDK-enabled applications in VM can cause SEGFAULT in host OpenVSwitch+DPDK

Hardware requirements

NFV features depend on hardware implementation. Therefore, you must verify that your hardware supports NFV features before running examples from this guide.

If you plan to enable SR-IOV, your environment must meet the following prerequisites:

- a physical network adapter must support SR-IOV
- a host system must support SR-IOV enabled on the device and I/O MMU (Intel VT-d or AMD-Vi)

If you plan to enable Huge Pages, your CPU must support Huge Pages.

If you plan to enable NUMA, your CPU must support NUMA.

If you plan to enable DPDK, your CPU must support:

- Streaming SIMD Extensions version 3 or higher
- pse or pdpe1gb

Limitations

This section describes the limitations of software packages used to enable NFV features.

For libvirt version before 1.2.12:

- LP1379346: Error creating a VM: internal error: No PCI buses available

For libvirt version 1.2.9.2:

- LP1449028: NUMA tuning broken in select libvirt versions

For QEMU version before 2.3.0:

- LP1417937: qemu appears to be built without CONFIG_NUMA defined

For DPDK library version before 16.04:

- LP1597004: DPDK-enabled applications in VM can cause SEGFAULT in host OpenVSwitch+DPDK

For ovs-dpdk-networking plugin:

- networking-ovs-dpdk bugs

For OpenStack:

- Guarantee resource allocation to NFV workloads. Extensive virtual CPU and RAM overprovisioning may result in latency spikes when several instances compete for the limited physical resources controlled by a hypervisor.

  Cloud administrators must verify that the corresponding OpenStack Compute service settings are configured to address this limitation in /etc/nova/nova.conf. For example:

  ```
  ram_allocation_ratio=1.0
  reserved_host_memory_mb=512
  cpu_allocation_ratio=1.0
  ```

Seealso

Nova configuration options

Prepare your environment for NFV

This section describes initialization steps needed to prepare your environment for enablement of NFV features.

Prepare DevStack environment

DevStack is a set of scripts used to quickly create an OpenStack development environment from the source in git repository master or specific branches suitable for development and operational testing. DevStack can be used to demonstrate running OpenStack services and provide examples of using them from a command line. For NFV enablement use DevStack running on Ubuntu 14.04.

To prepare DevStack environment:

1. If you deploy DevStack after installation of the Ubuntu packages, remove PPA of these packages:

   ```
   sudo add-apt-repository --remove ppa:ubuntu-cloud-archive/liberty-staging
   ```

   You need this because these packages are not authorized, and DevStack deployment will fail otherwise.
2. Install OS that supports NFV features you want to enable in an OpenStack environment.

   Note

   Any of the distributions mentioned in Prerequisites section will provide you with a suitable platform on top of which you can build your OpenStack environment.
3. Configure the interfaces needed for networking.
4. Disable any management entities such as Network Manager that might interfere with configuration. There are several ways this can be accomplished:
   - One way is to disable Network Manager and instead default to simple kernel network configuration by executing the following commands.
     1. Stop the NetworkManager service:

        ```
        systemctl stop NetworkManager
        ```
     2. Disable the NetworkManager service:

        ```
        systemctl disable NetworkManager
        ```

3. Make the network service start upon reboot:

   ```
   chkconfig --levels 345 network on
   ```

4. Start the network service:

   ```
   systemctl start network
   ```

- An alternative way is to update all configuration files for interfaces you wish to use with OpenStack that are located in the folder /etc/sysconfig/network-scripts/ by including the line:

  ```
  NM_CONTROLLED=no
  ```

  This will prevent Network Manager from changing interface settings.

5. Enable iptables and disable firewall.

> Note
>
> DevStack uses iptables rules. Therefore, DevStack will not work properly when the Linux firewall is enabled.

1. Save iptables rules:

   ```
   service iptables save
   ```

2. Disable firewall:

   ```
   systemctl disable firewalld
   ```

3. Enable iptables:

   ```
   systemctl enable iptables
   ```

4. Stop the firewall service:

   ```
   systemctl stop firewalld
   ```

5. Start iptables:

```
systemctl start iptables
```

6. Download DevStack:

   ```
   git clone https://git.openstack.org/openstack-dev/devstack
   ```

Seealso

- DevStack documentation

Verify packages versions

Your host environment for NUMA, DPDK, and Huge Pages must have the proper versions of libvirt and QEMU packages specified in Prerequisites.

To verify packages versions:

1. Verify libvirt version:

   ```
   libvirtd --version
   ```

   Example of system response

   ```
   libvirtd (libvirt) 1.2.16
   ```

2. Verify QEMU version:

   ```
   qemu-system-x86_64 -version
   ```

   Example of system response

   ```
   QEMU emulator version 2.3.0
   ```

Install packages for NUMA, DPDK, and Huge Pages

NUMA, DPDK, and Huge Pages require the following libvirt and QEMU packages installed on a host system:

- libvirt-dev
- libvirt-bin
- qemu-system-x86
- qemu-kvm

If required versions of libvirt and QEMU are not installed on the host machine, you need to install them manually.

To install libvirt and QEMU packages:

1. Log into the host machine.
2. Add PPA to the Ubuntu repository:

   ```
   sudo add-apt-repository ppa:ubuntu-cloud-archive/liberty-staging
   ```

   Warning

   If you deploy DevStack after installation of the provided packages, do not add PPA of these packages.

3. Retrieve new lists of packages:

   ```
   sudo apt-get update
   ```

4. Install the libvirt packages:

   ```
   sudo apt-get install libvirt-dev libvirt-bin
   ```

5. Install the QEMU packages:

   ```
   sudo apt-get install qemu-kvm qemu-system-x86
   ```

Enable NUMA

Introduction

This chapter describes how to enable and use NUMA on the OpenStack Liberty environment.

NUMA overview

Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors). The benefits of NUMA are limited to particular workloads, notably on servers where the data are often associated strongly with certain tasks or users.

NUMA prerequisites

This section decribes prerequisites for NUMA enablement.

Verify CPU support

CPU must support NUMA.

To verify NUMA support by CPU, run:

```
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 12 13 14 15 16 17
node 0 size: 128910 MB
node 0 free: 669 MB
node 1 cpus: 6 7 8 9 10 11 18 19 20 21 22 23
node 1 size: 129022 MB
node 1 free: 4014 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

Note

You can have one NUMA node. But, in practice, a good NUMA topology should contain at least two NUMA nodes.

Verify nested KVM support

The tests require support for nested KVM, which is not enabled by default on hosts with a hypervisor. You need to turn on the nested KVM support explicitly on the host when loading the kvm-intel or kvm-amd kernel modules.

To verify nested KVM support on Intel hosts, run:

```
cat /sys/module/kvm_intel/parameters/nested
```

Example of system response:

```
N
```

If the nested KVM support is not enabled, run:

```
rmmod kvm-intel
echo "options kvm-intel nested=y" > /etc/modprobe.d/dist.conf
modprobe kvm-intel

cat /sys/module/kvm_intel/parameters/nested
```

Example of system response:

```
Y
```

To verify nested KVM support on AMD hosts, run:

```
cat /sys/module/kvm_amd/parameters/nested
```

Example of system response:

```
0
```

If the nested KVM support is not enabled, run:

```
rmmod kvm-amd
echo "options kvm-amd nested=1" > /etc/modprobe.d/dist.conf
modprobe kvm-amd

cat /sys/module/kvm_amd/parameters/nested
```

Example of system response:

```
1
```

Install and configure NUMA environment

Emulate NUMA environment

If your CPU does not support NUMA, you can emulate NUMA environment for development and debugging purposes. However, you will not have any performance improvement in the case of emulation.

Mirantis recommends you to have at least two NUMA nodes. If the numactl -H command shows only one NUMA node, you can emulate NUMA nodes.

To emulate NUMA nodes:

1. Provision a basic Ubuntu Server 14.04 AMD64 guest with 4 virtual CPUs, 8 GB of RAM, and 50 GB of disk space:

   Warning

   Ensure virt-viewer is installed to finish Ubuntu installation.

   ```
   mkdir ~/images
   cd ~/images
   wget http://releases.ubuntu.com/14.04/ubuntu-14.04.3-server-amd64.iso
   virt-install --name ubuntu_14 --ram 8000 --vcpus 4 --file ~/images/ubuntu_14.img \
               --file-size 50 --cdrom ~/images/ubuntu-14.04.3-server-amd64.iso
   ```

2. Shutdown the VM:

   ```
   virsh destroy ubuntu_14
   ```

3. Open the VM's configuration file to edit:

```
virsh edit ubuntu_14
```

4. Create or replace the CPU section:

```
<cpu mode='host-passthrough'>
    <numa>
      <cell id='0' cpus='0-1' memory='4096000' unit='KiB'/>
      <cell id='1' cpus='2-3' memory='4096000' unit='KiB'/>
    </numa>
  </cpu>
```

   This example emulates two NUMA nodes; each NUMA node has 2 CPUs and 4GB RAM.

5. Start the VM:

```
virsh start ubuntu_14
```

6. Verify that the VM's CPU section has been changed:

```
virsh dumpxml ubuntu_14 | grep cpu
```

   Example of system response:

```
<vcpu placement='static'>4</vcpu>
<cpu mode='host-passthrough'>
    <cell cpus='0-1' memory='4096000'/>
    <cell cpus='2-3' memory='4096000'/>
</cpu>
```

7. Connect to the VM using SSH. Take all further steps on the VM.

Configure NUMA

To configure NUMA:

1. Add the following parameter to /etc/nova/nova.conf:

```
[libvirt]
virt_type = kvm
```

2. Add NUMATopologyFilter to scheduler_default_filters:

```
[DEFAULT]
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,ComputeCapabilitie
```

3. Restart the nova-scheduler and nova-compute services.

```
sudo service nova-scheduler restart
sudo service nova-compute restart
```

Example configurations

This section describes examples of using NUMA when booting VM in an OpenStack environment:

Example: Boot VM with two NUMA nodes

This example demonstrates booting VM with two NUMA nodes.

To boot VM with two NUMA nodes:

1. Create a new flavor or use an existing one to use with NUMA. To create a new flavor, run:

```
. openrc admin admin        # get admin rights
nova flavor-create m1.numa 999 1024 5 4
```

2. Add numa_nodes to the flavor.

> Note
>
> vCPUs and RAM will be divided equally between the NUMA nodes.

```
nova flavor-key m1.numa set hw:numa_nodes=2
nova flavor-show m1.numa
```

Example of system response:

```
+----------------------------+------------------------+
| Property                   | Value                  |
+----------------------------+------------------------+
| OS-FLV-DISABLED:disabled   | False                  |
| OS-FLV-EXT-DATA:ephemeral  | 0                      |
| disk                       | 5                      |
| extra_specs                | {"hw:numa_nodes": "2"} |
| id                         | 999                    |
| name                       | m1.numa                |
| os-flavor-access:is_public | True                   |
| ram                        | 1024                   |
| rxtx_factor                | 1.0                    |
| swap                       |                        |
| vcpus                      | 4                      |
+--------------------------+----------------------+
```

3. Create a new image or use an existing image.

> Note
>
> You need an Ubuntu image and the default Cirros image.

To create a new Ubuntu image, run:

```
glance --os-image-api-version 1 image-create --name ubuntu \
      --location https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
      --disk-format qcow2 --container-format bare
```

4. To enable SSH connections, do:

    1. Add a new rule to the security group:

    ```
    nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
    ```

    2. Create a new ssh key pair or use the existing key pair. To create a new ssh key pair, run:

    ```
    ssh-keygen
    ```

    3. Add the key pair to the Computing service:

    ```
    nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp
    ```

5. Verify free memory before booting the VM:

```
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 3856 MB
node 0 free: 718 MB
```

```
node 1 cpus: 2 3
node 1 size: 3937 MB
node 1 free: 337 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

6. Boot a new instance using the created flavor.

```
nova boot --flavor m1.numa --image ubuntu --key-name my_kp inst1
```

7. Verify if free memory has been changed after booting the VM:

```
numactl -H
```

   Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 3856 MB
node 0 free: 293 MB           # was 718 MB
node 1 cpus: 2 3
node 1 size: 3937 MB
node 1 free: 81 MB            # was 337 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

8. Retrieve the instance's IP:

```
nova show inst1 | awk '/network/ {print $5}'
```

   Example of system response:

```
10.0.0.2
```

9. Connect to the VM using SSH:

```
ssh ubuntu@10.0.0.2
```

10. Install numactl:

    ```
    sudo apt-get install numactl
    ```

11. Verify the NUMA topology on the VM:

    ```
    numactl -H
    ```

    Example of system response:

    ```
    available: 2 nodes (0-1)
    node 0 cpus: 0 1
    node 0 size: 489 MB
    node 0 free: 393 MB
    node 1 cpus: 2 3
    node 1 size: 503 MB
    node 1 free: 323 MB
    node distances:
    node   0   1
      0:  10  20
      1:  20  10
    ```

Example: Boot VM with CPU and memory pining

This example demonstrates booting VM with CPU and memory pining.

To boot VM with CPU and memory pining:

1. Create a new flavor with specific division of vCPUs and RAM between the NUMA nodes:

   ```
   . openrc admin admin        # get admin rights
   nova flavor-create m1.numa_2 9992 1024 5 4
   ```

2. Add numa_nodes and other specific options to the flavor:

   ```
   nova flavor-key m1.numa_2 set hw:numa_nodes=2 hw:numa_cpus.0=0,2 hw:numa_cpus.1=1,3 \
                               hw:numa_mem.0=324 hw:numa_mem.1=700
   nova flavor-show m1.numa_2 | grep extra
   ```

   Example of system response:

```
| extra_specs                | {"hw:numa_cpus.0": "0,2", "hw:numa_cpus.1": "1,3", "hw:numa_nodes": "2", "h
```

3. Create a new image or use an existing image.

> Note
>
> You need an Ubuntu image or the default Cirros image.

To create a new Ubuntu image, run:

```
glance --os-image-api-version 1 image-create --name ubuntu \
      --location https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
      --disk-format qcow2 --container-format bare
```

4. To enable SSH connections, do:

   1. Add a new rule to the security group:

   ```
   nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
   ```

   2. Create a new ssh key pair or use the existing key pair. To create a new ssh key pair, run:

   ```
   ssh-keygen
   ```

   3. Add the key pair to the Computing service:

   ```
   nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp
   ```

5. Boot a new instance using the created flavor.

```
nova boot --flavor m1.numa_2 --image ubuntu --key-name my_kp inst2
```

6. Verify if free memory has been changed after booting the VM:

```
numactl -H
```

Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 1
node 0 size: 3856 MB
node 0 free: 293 MB           # was 718 MB
node 1 cpus: 2 3
node 1 size: 3937 MB
node 1 free: 81 MB            # was 337 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

7. Retrieve the instance's IP:

```
nova show inst2 | awk '/network/ {print $5}'
```

   Example of system response:

```
10.0.0.3
```

8. Connect to the VM using SSH:

```
ssh ubuntu@10.0.0.3
```

9. Install numactl:

```
sudo apt-get install numactl
```

10. Verify the NUMA topology on the VM:

```
numactl -H
```

   Example of system response:

```
available: 2 nodes (0-1)
node 0 cpus: 0 2
node 0 size: 303 MB
node 0 free: 92 MB
node 1 cpus: 1 3
node 1 size: 689 MB
node 1 free: 629 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

You can see that the NUMA topology has two NUMA nodes. Total RAM size is about 1 GB, node-0 CPUs are 0 and 2, node-1 CPUs are 1 and 3, node-1 RAM is about 324 MB, node-2 RAM is about 700 as specified in the m1.numa_2 flavor.

Configure NUMA

To configure NUMA:

1. Add the following parameter to /etc/nova/nova.conf:

```
[libvirt]
virt_type = kvm
```

2. Add NUMATopologyFilter to scheduler_default_filters:

```
[DEFAULT]
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,ComputeCapabilitie
```

3. Restart the nova-scheduler and nova-compute services.

```
sudo service nova-scheduler restart
sudo service nova-compute restart
```

Enable OVS and DPDK

Introduction

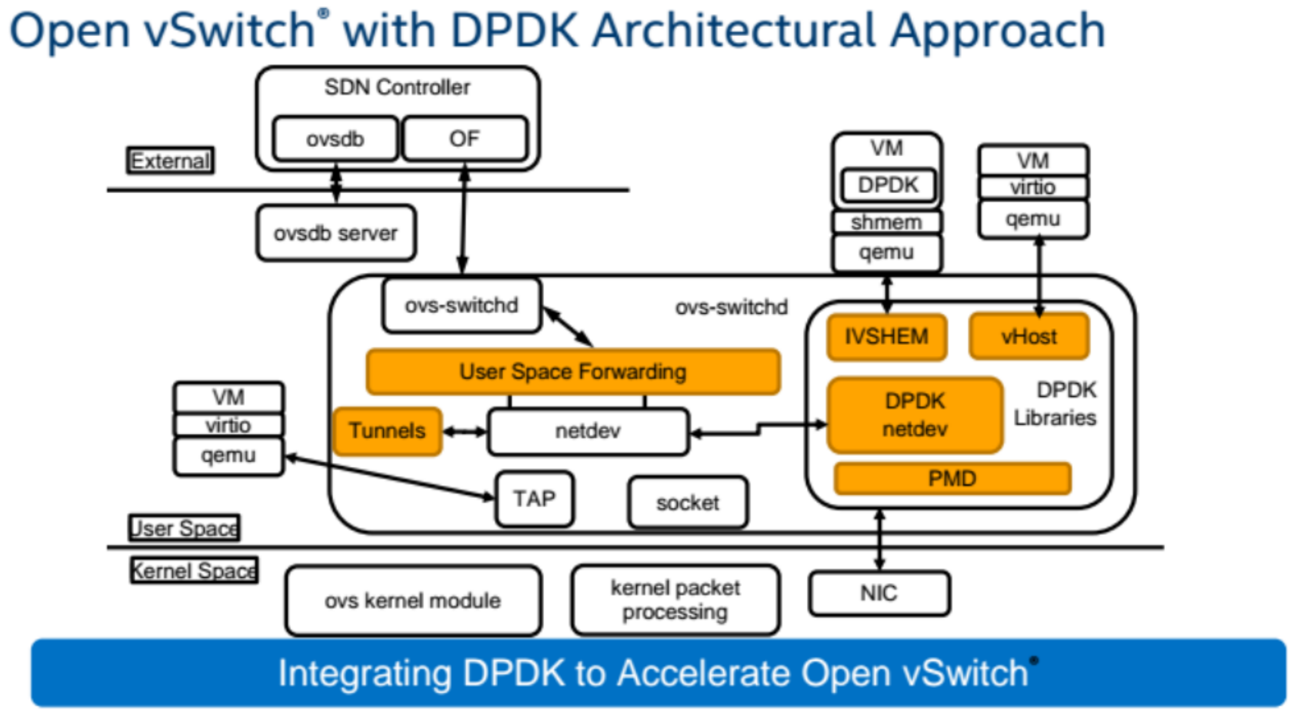
This section of the NFV guide explains the DPDK technology and points to the way you can enable it with DevStack.

Warning

OVS with DPDK support is an experimental feature.

DPDK Overview

Intel© DPDK is a set of libraries and drivers to perform fast packet processing in the user space. Open vSwitch uses DPDK functions and drivers as the netdev datapath to perform out-of-kernel packet processing. In this scenario, the VHOST_USER DPDK library will be used as the user space VHOST implementation. OpenStack users can perform fast host to guest to host packet processing with adding accelerated OVS to a compute node.



In the figure above, OVS uses DPDK libraries as the netdev datapath, poll mode drivers as the packet processing unit for host NICs and the vHost and ivshmem libraries as the host to guest control plane and data plane. Using the ivshmem library of the DPDK means running the DPDK application on a guest OS. In this scenario, the guest DPDK application must use host's shared Huge Pages, which is out of scope for this guide. For the VHOST_USER implementation, OVS provides socket devices for management of control messages and shared hugepages provides a mechanism for data plane packet processing.

Seealso

Related features of DPDK are described in http://dpdk.org/doc/guides/

Verify DPDK requirements

In most Fedora, Ubuntu, or RHEL distributions, you can use the vendor-supplied kernel configuration to run most of the DPDK applications.

To verify DPDK requirements:

1. Verify the kernel version:

   ```
   uname -r
   ```

2. Verify if the host OS supports UIO:

   ```
   modprobe uio
   ll /sys/class | grep uio
   ```

3. Verify if the host OS supports VFIO-PCI:

   ```
   modprobe vfio
   ll /sys/class | grep vfio
   ```

4. Verify if HUGETLBFS is enabled:

   ```
   egrep '\[always\]' /sys/kernel/mm/transparent_hugepage/enabled
   ```

   Example of system response:

   ```
   [always] madvise never
   ```

5. Verify if the host CPU supports hugepages:

   ```
   egrep '(pse|pdpe1gb)' /proc/cpuinfo
   ```

Example of system response:

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts a
```

> Note
>
> PSE stands for 2MB hugepages support and PDPE1GB - 1GB hugepages support.

6. Verify if CPU supports SSE instruction sets:

```
egrep 'sse' /proc/cpuinfo
```

Example of system response:

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts a
```

> Seealso
>
> The full list of requirements for all DPDK applications

Configure DPDK

This section describes configuration steps to enable OVS with DPDK support.

Enable OVS with DPDK ports in Neutron

You can enable OVS with DPDK ports in Neutron by installing the ovs-dpdk-networking plugin, which is a collection of agents and drivers needed to manage DPDK accelerated OVS with Neutron.

To enable OVS+DPDK ports in Neutron:

1. Enable the networking-ovs-dpdk plugin.
2. Within the DevStack directory create the local.conf file or modify the existing one in the following way:

```
enable_plugin networking-ovs-dpdk https://review.openstack.org/openstack/networking-ovs-dpdk maste
OVS_NUM_HUGEPAGES=3072 # set up the hugepages amount on the host
```

```
OVS_DATAPATH_TYPE=netdev # set up appropriate dapatath type (netdev used for DPDK application)
OVS_LOG_DIR=/opt/stack/logs # specify logging directory for OVS
```

3. To choose a role of the host, add several configuration parameters to local.conf.

> Note
>
> In Examples of configuration files section, at the end of this chapter, you can find two examples of how to create local.conf for controller and compute nodes.

4. Run the ./stack.sh script to download OpenStack components.

By enabling the networking-ovs-dpdk plugin the default OVS support will be overridden by the OVS+DPDK support. The DevStack init mechanics provide initial scripts within the plugin's directory.

Configure libvirt

DPDK needs hugepages enabled in libvirt.

To configure libvirt with Huge Pages:

1. Add a mount point for the hugelbfs file system to the /etc/libvirt/qemu.conf configuration file.

```
hugetlbfs_mount = "/run/hugepages/kvm"  # this is default mount point
```

Boot a virtual machine with Huge Pages

A kvm-based VM requires an appropriate flavor for booting that enables using hugepages as a vNIC's data plane back end.

To create a flavor and boot a VM, run:

1. Create a new flavor or use an existing one:

```
. openrc admin admin        # get admin rights
nova flavor-create huge 999 2048 4 2 # 999 1024 4 1 for default
```

2. Add size of huge pages to the flavor:

```
nova flavor-key huge set hw:mem_page_size=2048
nova flavor-show huge
```

Example of system response:

```
+----------------------------+------------------------------+
| Property                   | Value                        |
+----------------------------+------------------------------+
| OS-FLV-DISABLED:disabled   | False                        |
| OS-FLV-EXT-DATA:ephemeral  | 0                            |
| disk                       | 4                            |
| extra_specs                | {"hw:mem_page_size": "2048"} |
| id                         | 7                            |
| name                       | huge                         |
| os-flavor-access:is_public | True                         |
| ram                        | 1024                         |
| rxtx_factor                | 1.0                          |
| swap                       |                              |
| vcpus                      | 1                            |
+----------------------------+------------------------------+
```

3. Create a new image or use an existing glance image.

> Note
>
> Use the Ubuntu image from the Ubuntu images repository and the default Cirros image.

If needed, you can create an image named ubuntu with the following command:

```
glance --os-image-api-version 1 image-create --name ubuntu \
    --location https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
    --disk-format qcow2 --container-format bare
```

4. Boot instance with the created flavor:

```
nova boot --flavor huge --image ubuntu inst1
```

When booting, the Compute service creates the appropriate vhost_user port in the OVS integration bridge. To verify this, run:

```
sudo ovs-vsctl show
```

Example of system response:

```
796b6936-7d5f-4cad-917d-52bc78962c78
 Bridge br-int
    fail_mode: secure
    Port "tap1623aff5-33"
        tag: 1
        Interface "tap1623aff5-33"
            type: internal
    Port br-int
        Interface br-int
            type: internal
    Port "qr-ac082c22-3d"
        tag: 1
        Interface "qr-ac082c22-3d"
            type: internal
    Port "vhua1434f2e-fd"         #new port vhost-user type
        tag: 2
        Interface "vhua1434f2e-fd"
            type: dpdkvhostuser
    Port "qr-15a7bc43-5c"
        tag: 1
        Interface "qr-15a7bc43-5c"
            type: internal
    Port patch-tun
        Interface patch-tun
            type: patch
            options: {peer=patch-int}
 Bridge br-tun
    fail_mode: secure
    Port patch-int
        Interface patch-int
            type: patch
            options: {peer=patch-tun}
    Port br-tun
        Interface br-tun
            type: internal
 Bridge br-ex
    Port "qg-d77a0c60-9d"
        Interface "qg-d77a0c60-9d"
            type: internal
    Port br-ex
        Interface br-ex
            type: internal
```

5. Verify the VM's state after boot:

```
nova list
```

Example of system response:

```
+--------------------------------------+--------------+--------+------------+-------------+-----------------------------------+
| ID                                   | Name         | Status | Task State | Power State | Networks                          |
+--------------------------------------+--------------+--------+------------+-------------+-----------------------------------+
| 73280c74-f263-442e-a56a-0321da23b0c4 | inst1        | ACTIVE | -          | Running     | private1337=19
| 477a0c70-8b40-464e-853d-63a8ff50ad5a | inst2        | ACTIVE | -          | Running     | private1337=19
+--------------------------------------+--------------+--------+------------+-------------+-----------------------------------+
```

Seealso

Ubuntu images repository

Examples

This section demonstrates various CLI and GUI interactions that you can perform using the DPDK technology described above.

Example: Enable OVS with DPDK support on the same host

This example demonstrates launching two virtual machines on a single compute node, with the compute node using OVS with DPDK support. The example also shows communication between these virtual machines over the VM's single vNIC.

The following hardware node is used in the example:

- the host name cz7119
- running Intel Xeon 12 cores and 64 GB RAM
- OS installed Ubuntu 14.04 LTS Trusty Thar
- two NICs with 1GB/s each, where eth0 is a single external network interface and eth1 is an unused interface

To configure a single All-in-one node, use the configuration example from the Examples of configuration files section at the end of this chapter.

To enable OVS with DPDK support on the same host:

1. Create the local.conf file within the devstack directory or edit the existing one to add the appropriate configuration.
2. Configure the host following the instructions from the previous chapter.
3. After getting OpenStack deployed on the single node, configure public and private networks.

   To configure public and private networks, either use the Horizon web UI or host CLI:

   1. Login to the Horizon web UI using default admin/admin login and password.

2. After switching the project to admin, enter the Network page.
3. Create two networks: the private one called private1337 192.11.0.0/24 and the external public network called public1337 172.24.4/24.
4. In the Routers section create a router for internal networks.
5. In the Gateway section create a gateway for the external public network.

> Note
>
> Do not forget to add the private network to the router’s port.

To access VMs using SSH:

1. Create a key-pair:

   ```
   ssh-keygen
   nova keypair-add --pub_key ~/.ssh/id_rsa.pub my_kp
   ```

2. Create instances using Horizon web UI.
3. Go to the Instances pad and launch two VMs with the names ubuntu-cloud-image-1 and ubuntu-cloud-image-2.

   > Note
   >
   > To work with vhost-user ports, VMs must use hugepage-configured flavor.
   >
   > Do not pre-configure the image for this. Use Ubuntu Cloud Image as mentioned in the previous chapter.

4. On the Security tab choose my_kp as a key-pair.
5. On the networking tab choose private network to allocate private IP addresses and click Launch to create VMs.
6. After getting VMs in the Active state, associate floating IPs from the public network. To do this, click Manage Instance and Associate floating IP.
7. Choose one of the IPs in the range. To do this, allocate IPs from the pool or take unused ones.
8. To access the external network, add appropriate NAT rules on the controller. For example, if eth0 is an API interface of the node, run:

   ```
   sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
   ```

9. Connect through SSH from the host with the key-pair to VMs directly.

   > Note
   >
   > If you use Ubuntu Cloud Image for a guest OS, the default user name is ubuntu.

   For example, to login to the VM with floating 172.24.4.4, run:

   ```
   ssh ubuntu@172.24.4.4
   ```

10. To verify that the guest interfaces have been properly configured, run ifconfig to check the network interfaces.

    1. Run ifconfig on ubuntu-cloud-image-1:

       ```
       ifconfig
       ```

       Example of system response:

       ```
       eth0      Link encap:Ethernet  HWaddr fa:16:3e:ce:e3:58
                 inet addr:192.11.0.5  Bcast:192.11.0.255  Mask:255.255.255.0
                 inet6 addr: fe80::f816:3eff:fece:e358/64 Scope:Link
                 UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                 RX packets:106 errors:0 dropped:0 overruns:0 frame:0
                 TX packets:111 errors:0 dropped:0 overruns:0 carrier:0
                 collisions:0 txqueuelen:1000
                 RX bytes:22859 (22.8 KB)  TX bytes:14150 (14.1 KB)

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

    2. Run ifconfig on ubuntu-cloud-image-2:

       ```
       ifconfig
       ```

       Example of system response:

```
eth0      Link encap:Ethernet  HWaddr fa:16:3e:ce:e3:58
          inet addr:192.11.0.6  Bcast:192.11.0.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:feef:1c3f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:107 errors:0 dropped:0 overruns:0 frame:0
          TX packets:101 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23057 (23.0 KB)  TX bytes:13130 (13.1 KB)

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

To run a simple test:

Ping one VM from another:

```
ping 192.11.0.6
```

Example of system response:

```
PING 192.11.0.6 (192.11.0.6) 56(84) bytes of data.
64 bytes from 192.11.0.6: icmp_seq=1 ttl=64 time=0.414 ms
64 bytes from 192.11.0.6: icmp_seq=2 ttl=64 time=0.152 ms
64 bytes from 192.11.0.6: icmp_seq=3 ttl=64 time=0.160 ms
64 bytes from 192.11.0.6: icmp_seq=4 ttl=64 time=0.154 ms
...
```

To measure traffic performance:

1. Install iperf:

   ```
   sudo apt-get install iperf
   ```

2. Run iperf as a server on ubuntu-cloud-image-1:

   ```
   iperf -s
   ```

3. Run iperf as a client on ubuntu-cloud-image-2:

   ```
   iperf -c 192.11.0.5
   ```

   where 192.11.0.5 is the private IP address of ubuntu-cloud-image-1.

4. Verify iperf test results on ubuntu-cloud-image-1:

   ```
   ------------------------------------------------------------
   Server listening on TCP port 5001
   TCP window size: 85.3 KByte (default)
   ------------------------------------------------------------
   [  4] local 192.11.0.5 port 5001 connected with 192.11.0.6 port 41230
   [ ID] Interval       Transfer     Bandwidth
   [  4]  0.0-10.0 sec  1.09 GBytes  939 Mbits/sec
   ```

Example: Enable OVS with DPDK support on different hosts

This example demonstrates launching two virtual machines on different compute nodes, with both compute nodes using OVS with DPDK support.

The following hardware nodes are used in the example:

- hostnames cz7119 and cz7120 with 2x1GB interfaces each
- nodes have Intel Xeon and 64 GB RAM
- hosts OS: Ubuntu Trusty 14.04, Linux kernel version 3.13

Running a single node test significantly differs from a multi-node test in the network topology. To run this example, you need the All-in-one node with the same configuration as in the previous example and a compute node with local.conf from the Examples of configuration files section at the end of this chapter.

To enable OVS with DPDK support on different hosts:

1. Set up extra interfaces called data interfaces. Each node has an interface connected to another interface through a switch.
2. With the selected VLAN tenant segmentation you need to open switch ports for a configured range of VLANs.
3. Set up appropriate local.conf files.
4. Run the devstack initialization script.

```
./stack.sh
```

During the DevStack configuration, the networking-ovs-dpdk driver binds data interface on each node to the appropriate DPDK driver and adds the appropriate interface as the DPDK port to OVS. The tenant's traffic between VMs will be managed by data interfaces.

Setting up of the VMs for this example is almost the same. The only difference is that you need to verify if the Compute service spawned VMs on the different nodes.

To verify if VMs are spawned on the different nodes:

1. Go to the Admin tab.
2. View the Instances page. The Instances page lists all VMs of all projects with the associated hosts.

   In this example, ubuntu-cloud-image-1 is running on the host named cz7119 and ubuntu-cloud-image-2 is running on the cz7120 host.
3. Follow the previous example to setup two VMs from the Horizon tab.

> Note
>
> You do not need to do extra work to setup the connectivity between two VMs.

Examples of configuration files

This section contains examples of configuration files for a compute node that you can use in the deployment scenarios provided.

Example of the configuration file for a compute node

local.conf:

```
[[local|localrc]]

ADMIN_PASSWORD=admin
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

HOST_IP_IFACE=eth0
HOST_IP=<API_IP_ADDRESS>
HOST_NAME=$(hostname)

SERVICE_HOST=<ALL_IN_ONE_API_IP_ADDRESS>

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOST=$SERVICE_HOST
KEYSTONE_AUTH_HOST=$SERVICE_HOST
```

```
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

HORIZON_PASSWORD=admin

enable_plugin networking-ovs-dpdk https://review.openstack.org/openstack/networking-ovs-dpdk master
OVS_DPDK_MODE=compute
disable_all_services
enable_service n-cpu
enable_service q-agt

DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/xstack.sh.log
LOGDAYS=1

#Dual socket platform with 16GB RAM,3072*2048kB hugepages leaves ~4G for the system.
OVS_NUM_HUGEPAGES=3072

#Dual socket platform with 64GB RAM,14336*2048kB hugepages leaves ~6G for the system.
#OVS_NUM_HUGEPAGES=14336

OVS_DATAPATH_TYPE=netdev
OVS_LOG_DIR=/opt/stack/logs
OVS_SOCKET_MEM=1024,0                        #socket memory amount
OVS_HUGEPAGE_MOUNT=/run/hugepages/kvm             #hugepages mount point
OVS_BRIDGE_MAPPINGS=default:br-eth1

Q_PLUGIN=ml2
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,logger
Q_AGENT=openvswitch
Q_ML2_TENANT_NETWORK_TYPE=vlan
ML2_VLAN_RANGES=default:70-79


MULTI_HOST=1

[[post-config|$NOVA_CONF]]
[DEFAULT]
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=$HOST_IP
```

Example of the configuration file for All-in-one host machine

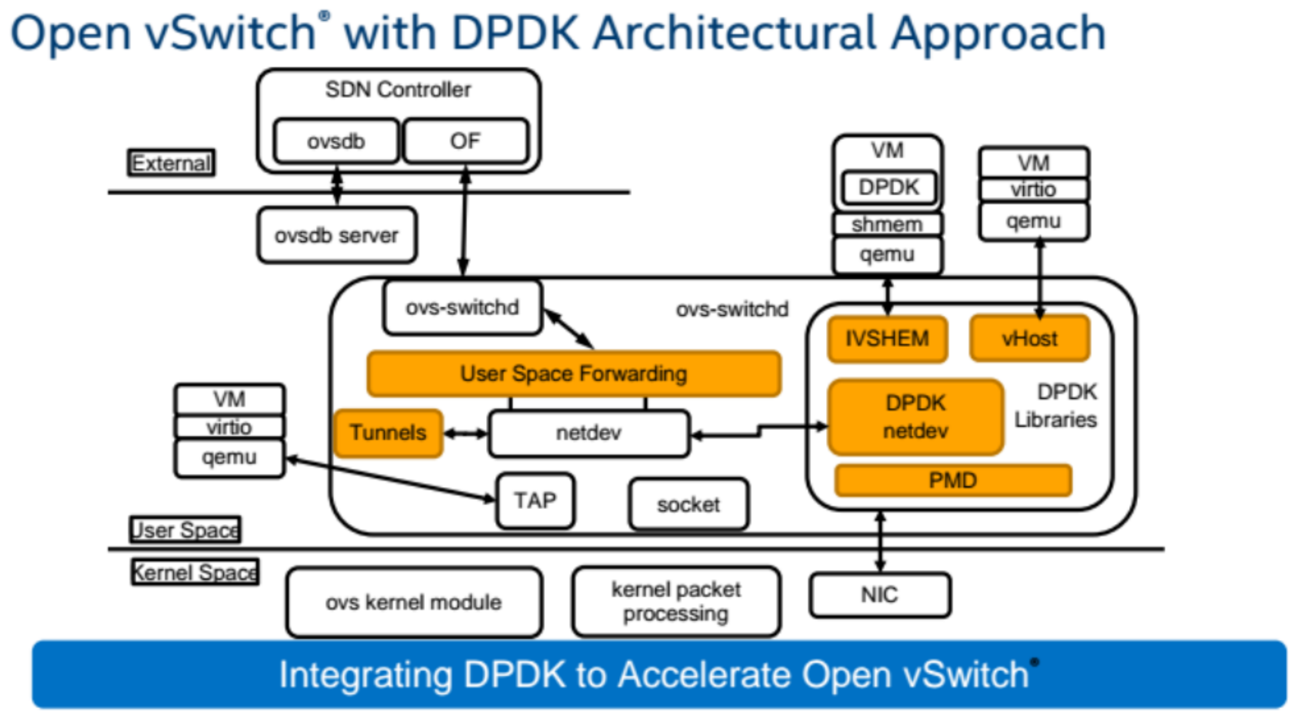
local.conf:

```
[[local|localrc]]

ADMIN_PASSWORD=admin
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

HOST_IP_IFACE=eth0
HOST_IP=<API_IP_ADDRESS>
HOST_NAME=$(hostname)


HORIZON_PASSWORD=admin

enable_plugin networking-ovs-dpdk https://review.openstack.org/openstack/networking-ovs-dpdk master
OVS_DPDK_MODE=controller_ovs_dpdk

disable_service n-net
disable_service tempest
enable_service n-cpu
enable_service q-agt
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron
enable_service q-vpn
enable_service q-fwaas

Q_PLUGIN=ml2
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,logger
Q_AGENT=openvswitch
Q_ML2_TENANT_NETWORK_TYPE=vlan
ML2_VLAN_RANGES=default:70:79

DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen
LOGFILE=${SCREEN_LOGDIR}/xstack.sh.log
LOGDAYS=1

#Dual socket platform with 16GB RAM,3072*2048kB hugepages leaves ~4G for the system.
OVS_NUM_HUGEPAGES=3072

#Dual socket platform with 64GB RAM,14336*2048kB hugepages leaves ~6G for the system.
#OVS_NUM_HUGEPAGES=14336
```

```
OVS_DATAPATH_TYPE=netdev
OVS_LOG_DIR=/opt/stack/logs
OVS_SOCKET_MEM=1024,0                          #socket memory amount
OVS_HUGEPAGE_MOUNT=/run/hugepages/kvm               #hugepages mount point
OVS_BRIDGE_MAPPINGS=default:br-eth1
```

DPDK Overview

Intel© DPDK is a set of libraries and drivers to perform fast packet processing in the user space. Open vSwitch uses DPDK functions and drivers as the netdev datapath to perform out-of-kernel packet processing. In this scenario, the VHOST_USER DPDK library will be used as the user space VHOST implementation. OpenStack users can perform fast host to guest to host packet processing with adding accelerated OVS to a compute node.



In the figure above, OVS uses DPDK libraries as the netdev datapath, poll mode drivers as the packet processing unit for host NICs and the vHost and ivshmem libraries as the host to guest control plane and data plane. Using the ivshmem library of the DPDK means running the DPDK application on a guest OS. In this scenario, the guest DPDK application must use host's shared Huge Pages, which is out of scope for this guide. For the VHOST_USER implementation, OVS provides socket devices for management of control messages and shared hugepages provides a mechanism for data plane packet processing.

Seealso

Related features of DPDK are described in http://dpdk.org/doc/guides/

Enable SR-IOV

Introduction

This chapter describes the process of using Single-Root I/O Virtualization (SR-IOV) to accelerate data plane performance for virtual machines running on the OpenStack platform.

You can find instructions for configuring and installing the required libraries, dependencies, and applications along with various constraints that Open vSwitch (OVS) with SR-IOV has in comparison with regular OVS installations.

The chapter explains data plane communication paths between virtual machines running on compute hosts with and without SR-IOV.

Warning

SR-IOV support is an experimental feature.

SR-IOV overview

Modern SR-IOV is a combination of several different technologies, of which PCI Express (PCIe) SR-IOV is one piece.

The first component required to enable SR-IOV is an input/output memory management unit (I/O MMU), which prevents virtual functions (VFs) from accessing memory regions unrelated to their given task. I/O MMU connects a direct memory access-capable (DMA-capable) I/O bus to the main memory. I/O MMU will map virtual DMA addresses to physical memory addresses on the system. Similar to how a memory management unit (MMU) manages an existing memory, I/O MMU can provide memory protection and reports attempts to access a memory that should not be accessed by a device.

The second component needed to use SR-IOV for KVM virtualization is the ability to assign PCI ports to virtual guests referred to as PCI device passthrough. PCI passthrough makes devices accessible as if they were a part of a guest itself. As a result, devices can achieve near bare-metal performance for networking tasks, since they can bypass much of the overhead involved in passing traffic through the host.

The final component is SR-IOV functionality itself. SR-IOV enables the partitioning of a single physical function (PF) into multiple virtual functions (VFs). The advantage of using VFs is that each VF can then be assigned to separate tasks. By making use of I/O MMU and PCI device passthrough, you can split up a single PF into a number of virtual functions. Thus, resources, such as access to physical networks, can be shared while still providing isolation and security of virtualization.

SR-IOV requirements

This section describes requirements for SR-IOV enablement.

To assign a virtual function to a guest, you need to satisfy the following conditions:

1. You need to have an SR-IOV capable device. To identify if a device supports SR-IOV, check for an SR-IOV capability in the device configuration. The device configuration also contains the number of VFs the device can support. The example below shows a simple test to determine if the device (Intel X540-T2 network adapter) located at the 1:00.0 (bus:device.function number) can support SR-IOV.

   ```
   lspci -vvv -s 1:00.0 | grep -A 9 SR-IOV | cut 2-72
   ```

   Example of system response:

   ```
   Capabilities: [160 v1] Single Root I/O Virtualization (SR-IOV)
       IOVCap:    Migration-, Interrupt Message Number: 000
       IOVCtl:    Enable- Migration- Interrupt- MSE- ARIHierarchy+
       IOVSta:    Migration-
       Initial VFs: 64, Total VFs: 64, Number of VFs: 0, Function Dependency
       VF offset: 128, stride: 2, Device ID: 10ed
       Supported Page Size: 00000553, System Page Size: 00000001
       Region 0: Memory at 0000000000000000 (64-bit, non-prefetchable)
       Region 3: Memory at 0000000000000000 (64-bit, non-prefetchable)
       VF Migration: offset: 00000000, BIR: 0
   ```

2. You need to have a host system that supports SR-IOV enabled on the device. Normally, such systems support I/O MMU.

   Note

   Intel provides I/O MMU feature under the name Intel VT-d, and in the case of AMD the feature is called AMD-Vi.

3. BIOS should support populating the base address registers that the SR-IOV extended configuration space contains. If the base address registers, regions 0 and 3 above, for the VFs are not populated and read as 0 as in the example, you can add a kernel parameter pci=realloc that may correct this issue on the next boot.

4. The kernel should support SR-IOV. Most modern kernels should have support for the feature. But you must enable SR-IOV support in the kernel configuration when you build the kernel.

   The Linux kernel has supported:

   - SR-IOV since the version 2.6.30
   - the configuration of the MAC address and VLAN of a VF since the version 2.6.34
   - dynamic enablement of SR-IOV through the sysfs sriov_numvfs value since the version 3.8

5. A distribution that supports SR-IOV. Most modern distributions support SR-IOV. This documentation contains examples using the CentOS 7 distribution. In general, for DevStack you can use Ubuntu 14.04, Fedora 22 or newer, or CentOS/RHEL 7.

Enable I/O MMU

If your hardware and software satisfy the requirements above, you can enable I/O MMU.

To enable the Intel I/O MMU, add the following parameters to the command line that invokes the kernel:

```
iommu=pt
intel_iommu=on
```

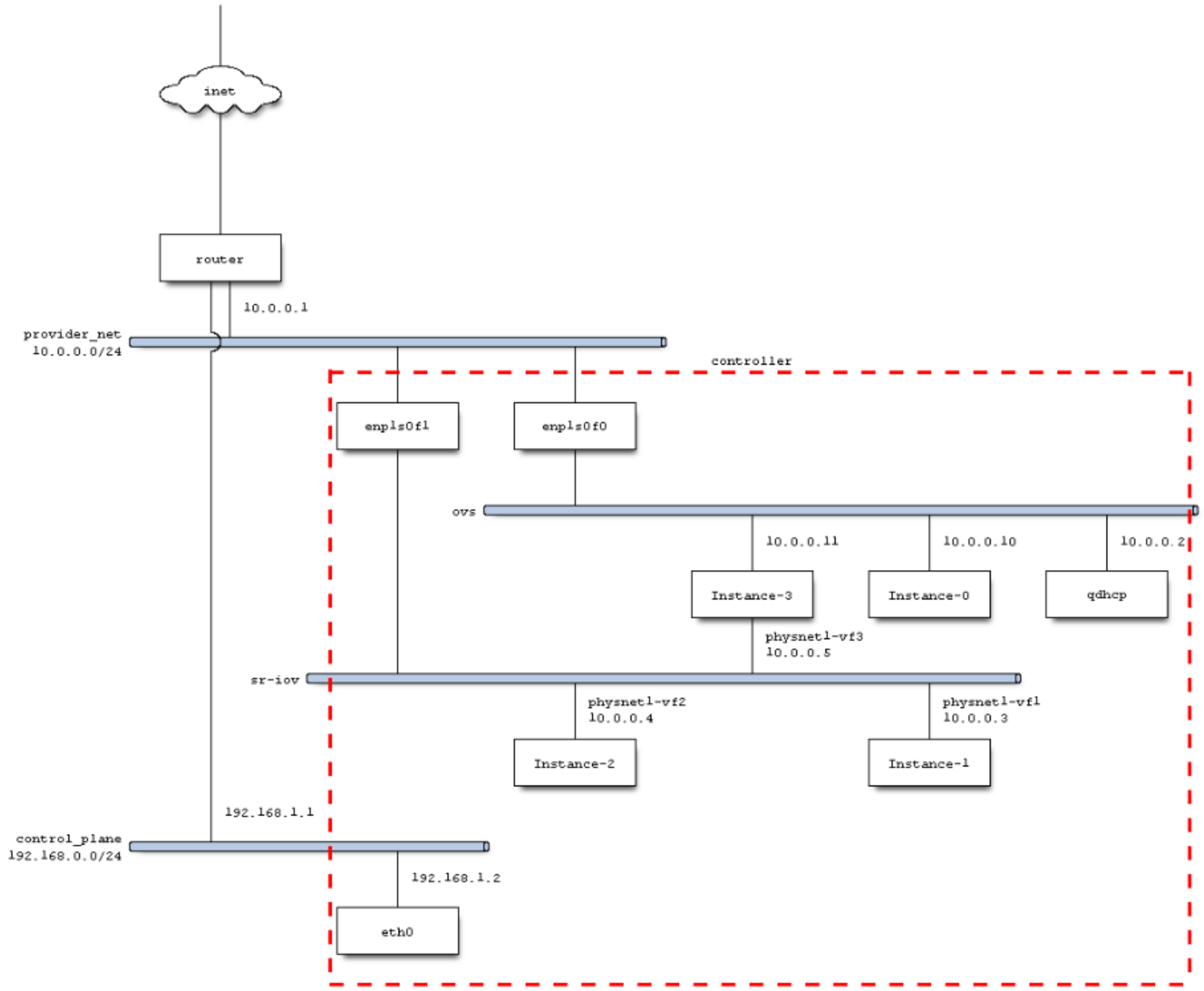
- iommu=pt enables a feature called identity mapping. By identity mapping of the host memory to the device, the host accesses I/O MMU at much lower cost.
- intel_iommu=on enables Intel I/O MMU. In some cases, this argument can be omitted. For kernel configurations where I/O MMU is not enabled by default due to performance impacts, you can mitigate these impacts with iommu=pt.

Configure SR-IOV OpenStack environment

This section provides an example of the OpenStack environment with SR-IOV support.

In this example, the node uses the following network interfaces:

- eth0 (00:19.0) - provides a host IP for management on the network 192.168.1.0/24.
- enp1s0f0 (01:00.0) - connects guests to the physical network 10.0.0.0/24. This is a trunk interface between OVS and the physical network.
- enp1s0f1 (01:00.1) - also connects guests to the physical network 10.0.0.0/24. The interface provides SR-IOV connectivity.



Warning

Do not use OVS on the interface that provides SR-IOV connectivity as most NICs contain only a simple switch that mirrors the external port at best when promiscuous mode is enabled. As a result, some vendors disable promiscuous functionality when SR-IOV is enabled. This, in turn, disables the ability to trunk VLANs, which OVS requires.

To configure SR-IOV OpenStack environment:

1. Set up the port enp1s0f0 in a promiscuous mode to act as a VLAN trunk for OVS.
2. Set up the port enp1s0f1 with VLAN to access the provider network, which you can define later:

```
# Bring up public interface and PF
ip link set dev enp1s0f0 up
ip link set dev enp1s0f1 up
```

3. Allocate the VFs:

```
# Attempt to enable SR-IOV with 7 VFs
echo 7 > /sys/class/net/enp1s0f1/device/sriov_numvfs
```

4. Set up DevStack.

   1. Configure the local.conf file for DevStack. This file defines much of the layout and topology for the environment. The example uses Networking with OVS and provider networks.

```
[[local|localrc]]
HOST_IP=192.168.1.2
ADMIN_PASSWORD=nova
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

# Services
disable_service n-net
disable_service zookeeper
ENABLED_SERVICES+=,q-svc,q-dhcp,q-meta,q-agt,q-sriov-agt

## Neutron Options
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
TENANT_VLAN_RANGE=3001:4000
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-enp1s0f0
PUBLIC_INTERFACE=enp1s0f0
Q_USE_PROVIDER_NETWORKING=True
Q_L3_ENABLED=False
IP_VERSION=4

## Neutron Networking options used to create Neutron Subnets
PROVIDER_NETWORK_TYPE="vlan"
SEGMENTATION_ID=2010

# ML2 Configuration
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,sriovnicswitch
Q_ML2_PLUGIN_TYPE_DRIVERS=vlan,flat,local
Q_ML2_TENANT_NETWORK_TYPE=vlan
```

```
# ML2 SR-IOV agent configuration
enable_plugin neutron git://git.openstack.org/openstack/neutron.git
PHYSICAL_DEVICE_MAPPINGS=physnet1:enp1s0f1

# Default Fedora 23 image
IMAGE_URLS+="https://download.fedoraproject.org/pub/fedora/linux/releases/23/Cloud/x86_64/Ima

# Add PCI Passthru filter, add alias, add all ports on PF
[[post-config|$NOVA_CONF]]
[DEFAULT]
scheduler_default_filters=RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,
pci_alias={\\"name\\":\\"x540vf\\",\\"product_id\\":\\"1515\\",\\"vendor_id\\":\\"8086\\"}
pci_passthrough_whitelist={\\"devname\\":\\"enp1s0f1\\",\\"physical_network\\":\\"physnet1\\"}

# ML2 plugin bits for SR-IOV enablement of Intel x540 NIC
[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ml2_sriov]
supported_pci_vendor_devs = 8086:1528, 8086:1515
```

2. Create a script called local.sh that will take care of any DevStack's post-installation options. Below you can find a script that takes care of providing an ssh key for login, setting up some initial network rules to allow ssh and ping across the network, and creating seven initial VFs under the names physnet1-vf<i>, where <i> represents a value from 1 to 7.

```
#!/bin/bash

# Add default key for admin and demo
nova keypair-add --pub_key=/opt/stack/.ssh/id_rsa.pub stack-ssh

# Enable ping and ssh
for i in admin demo
do
  nova --os-project-name $i secgroup-add-rule default \
       tcp 22 22 0.0.0.0/0
  nova --os-project-name $i secgroup-add-rule default \
       icmp -1 -1 0.0.0.0/0
done

# Add host nameserver to provider_net
ns=`grep nameserver /etc/resolv.conf | head -n1 | awk '{print $2}'`
neutron --os-project-name demo subnet-update \
        --dns-nameserver $ns provider_net

# Create 3 VF ports
for i in `seq 1 3`
do
  neutron --os-project-name demo port-create physnet1 \
        --vnic-type direct --name physnet1-vf$i
done
```

3. Start installation:

```
./stack.sh
```

DevStack installation may take half an hour or more. Once completed, you have an environment to begin creating instances that contain network devices routing either through the OVS network or a VF.

> Seealso
>
> The DevStack configuration is based on the example available at http://docs.openstack.org/developer/devstack/guides/neutron.html

Examples

This section describes examples of using SR-IOV in OpenStack environment with a mixed network consisting of instances using OVS connected vNIC and/or an SR-IOV interface.

Before running any of the following commands, set several environment variables to configure the user and project. For these scenarios, use the admin user and the demo project.

To configure the environment with the admin user and the demo project, run:

```
./openrc admin demo
```

Example: one instance with OVS vNIC and VLAN segmentation

This example demonstrates launching a single instance that contains a single vNIC that is running across OVS in a VLAN segmented environment.

To create the instance with OVS vNIC and VLAN segmentation, run:

```
# Get ID for physnet to later create OVS ports
net_id=`neutron net-show physnet1 -F id -f value`
# One instance, with one vNIC port on OVS w/ VLAN
nova boot --flavor m1.small --key_name stack-ssh \
        --image Fedora-Cloud-Base-23-20151030.x86_64 \
        --nic net-id=$net_id Instance-0
```

Note

You need to run the local.sh script listed in the Configure SR-IOV OpenStack environment section first, which creates physnet1-vf1 VF as well as the stack-ssh ssh key.

Example: two instances each with one SR-IOV VF

This example demonstrates launching two instances with each instance containing a single SR-IOV VF.

To create the instances each with one SR-IOV VF, run:

```
# Two instances, each with one VF port
for i in 1 2
  do
    port_id=`neutron port-show physnet1-vf$i -F id -f value`
    nova boot --flavor m1.small --key_name stack-ssh \
              --image Fedora-Cloud-Base-23-20151030.x86_64 \
              --nic port-id=$port_id Instance-$i
  done
```

Note

You need to run the local.sh script listed in the Configure SR-IOV OpenStack environment first, which creates physnet1-vf[1-2] VFs as well as the stack-ssh ssh key.

Example: one instance with both OVS vNIC and SR-IOV VF

This example demonstrates launching one instance with both an OVS vNIC and an SR-IOV VF.

To create the instance with both OVS vNIC and SR-IOV VF, run the script:

```
# Get ID for physnet to later create OVS ports
net_id=`neutron net-show physnet1 -F id -f value`
# One instance, with one VF port and one vNIC port on OVS w/ VLAN
for i in 3
  do
    port_id=`neutron port-show physnet1-vf$i -F id -f value`

     # determine the OVS port ID, and IP of VF
    port_ip=`neutron port-show physnet1-vf$i -F fixed_ips -f value | \
             sed 's/.* "//g' | sed 's/"}//g'`
    ovs_port_id=`neutron port-create physnet1 \
                         --allowed_address_pairs type=dict list=true \
                    ip_address=$port_ip | \
              grep " id" | awk '{print $4}'`

    # Create instance
    nova boot --flavor m1.small --key_name stack-ssh \
            --image Fedora-Cloud-Base-23-20151030.x86_64 \
            --user-data bond-config.sh \
            --nic port-id=$ovs_port_id \
            --nic port-id=$port_id Instance-$i
done
```

Note

You need to run the local.sh script listed in the Configure SR-IOV OpenStack environment section first, which creates physnet1-vf[1-3] VFs as well as the stack-ssh ssh key.

In addition, you can modify the OVS vNIC port to allow it to use the IP address belonging to the SR-IOV VF. By doing this, you can provide failover support for the VF in case of VM migration or if the link to the PF were to fail.

To enable the failover functionality from inside the VM, pass and run the script below to the guest as bond-config.sh. The script creates a bond of the VF and vNIC:

```
#!/bin/bash
cat << EOF > /etc/modprobe.d/bonding.conf
alias bond0 bonding
options bonding mode=active-backup primary=eth1
options bonding arp_ip_target=10.0.0.2 arp_interval=1000 fail_over_mac=active
EOF
cat << EOF > /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BOOTPROTO=dhcp
ONBOOT=yes
USERCTL=no
EOF
for i in eth0 eth1
do
cat << EOF > /etc/sysconfig/network-scripts/ifcfg-$i
DEVICE=$i
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
MASTER=bond0
SLAVE=yes
EOF
```

```
done
systemctl stop network
modprobe bonding
ifconfig bond0 up
ifenslave bond0 eth1
ifenslave bond0 eth0
systemctl restart network
```

Test connectivity

If a network provider does not have a router, instances can not access the Internet by default.

The script below makes use of the PF interface and provides NAT functionality so that the instances can access the Internet when an actual router is not available on VLAN 2010.

To test connectivity, enable access to the Internet from instances by running the script below:

```
#!/bin/bash
set -x

PUB=`route | grep default | awk '{print $NF}'`
PRIV=provider_net-gw

# Configure VLAN on PF so that we can access VFs without need of namespace
modprobe 8021q
ip link add link enp1s0f1 name $PRIV type vlan id 2010
ip addr add 10.0.0.1/24 dev $PRIV
ip link set dev $PRIV up

# Enable IP forwarding
sysctl -w net.ipv4.ip_forward=1
sysctl -w net.ipv4.conf.all.rp_filter=0
sysctl -w net.ipv4.conf.default.rp_filter=0

# Test for rules, if present then just exit
iptables -C FORWARD -i $PRIV -o $PUB -j ACCEPT && exit

# Add NAT rules
iptables -t nat -A POSTROUTING -o $PUB -j MASQUERADE
iptables -A FORWARD -i $PUB -o $PRIV -m state \
        --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -o $PUB -i $PRIV -j ACCEPT

# Move reject rule to the end of the list
iptables -A FORWARD -j REJECT --reject-with icmp-host-prohibited
iptables -D FORWARD -j REJECT --reject-with icmp-host-prohibited
```

Now, you can ping and ssh into the VMs. Also, with the NAT configured you can use utilities that require Internet access such as dnf to download and install any additional packages that you may want for guests to perform additional testing.

Install SR-IOV OpenStack environment

This section decribes the installation of an OpenStack environment with SR-IOV support.

To install your SR-IOV OpenStack environment:

1. Install OS that supports SR-IOV.

   Note

   Any of the distributions mentioned in Prerequisites section will provide you with a suitable platform on top of which you can build your OpenStack environment.

2. Configure the interfaces needed for networking.
3. Disable any management entities such as Network Manager that might interfere with configuration.

   There are several ways this can be accomplished:

   - One way is to disable Network Manager and instead default to simple kernel network configuration by executing the following commands.

     1. Stop the NetworkManager service:

        ```
        systemctl stop NetworkManager
        ```

     2. Disable the NetworkManager service:

        ```
        systemctl disable NetworkManager
        ```

     3. Start the network service upon reboot:

        ```
        chkconfig --levels 345 network on
        ```

     4. Start the network service:

        ```
        systemctl start network
        ```

- An alternative way is to update all configuration files for interfaces you wish to use with OpenStack that are located in the folder /etc/sysconfig/network-scripts/ by including the line:

  ```
  NM_CONTROLLED=no
  ```

  This will prevent Network Manager from changing interface settings.

4. Enable iptables and disable firewall.

   Note

   DevStack uses iptables rules. Therefore, DevStack will not work properly when the Linux firewall is enabled.

   1. Save iptables rules:

      ```
      service iptables save
      ```

   2. Disable firewall:

      ```
      systemctl disable firewalld
      ```

   3. Enable iptables:

      ```
      systemctl enable iptables
      ```

   4. Stop the firewall service:

      ```
      systemctl stop firewalld
      ```

   5. Start iptables:

      ```
      systemctl start iptables
      ```

5. Download DevStack:

   ```
   git clone https://git.openstack.org/openstack-dev/devstack
   ```

Enable Huge Pages

Introduction

This chapter describes how to enable and use Huge Pages in an OpenStack environment.

Huge Pages overview

When a process uses memory, CPU marks it as used by that process. For efficiency, CPU allocates RAM in chunks of 4K bytes, which is the default value on many platforms, called pages. Those pages can be swapped to disk. Since a process address space is virtual, the CPU and operating system have to remember which page belongs to which process, and where it is stored.

Access time to data stored in memory depends on the number of pages. The big number of pages slows down a data acquisition process.

For example, 1GB of memory used by a process has 262144 entries to look up (1GB / 4K). Every Page Table Entry is 8 bytes in size. The total number of bytes to look up is 2 Mb (262144 * 8).

Modern CPU architectures support bigger pages called Huge pages on Linux, Super Pages on BSD, or Large Pages on Windows. Huge pages help to minimize the number of entries to look up by CPU.

Verify CPU support

CPU must support Huge Pages as listed in Prerequisites.

To verify CPU support:

1. Log into the host machine.
2. Verify the CPU support for Huge Pages by typing:

   ```
   cat /proc/cpuinfo
   ```

   In the output, look for the following parameters:

   - PSE - support of 2MB hugepages
   - PDPE1GB - support of 1GB hugepages

Configure Huge Pages

The verified OpenStack environment with Huge Pages support needs the recommended versions of software listed in Prerequisites.

For testing, you need to reserve 1024 huge pages. Size of each page is 2 MB. The booted VM uses 512 of these pages (1GB RAM).

To configure huge pages, follow the steps:

1. Get ID of the kvm group on your host:

   ```
   grep kvm /etc/group
   ```

   Example of system response:

   ```
   kvm:x:111:
   ```

2. Add the count of huge pages you want to reserve and the ID of kvm group to /etc/sysctl.conf:

   ```
   vm.nr_hugepages = 1024  # 1024 pages. The size of each page equals 2 MB
   vm.hugetlb_shm_group = 111 # ‘kvm’ group id from step 2.
   ```

3. Apply changes:

   ```
   sudo sysctl -p
   ```

4. Verify that options have been applied:

   ```
   sudo sysctl -a | grep huge
   ```

   Example of system response:

   ```
   vm.hugepages_treat_as_movable = 0
   vm.hugetlb_shm_group = 111          # here
   vm.nr_hugepages = 1024              # and here
   vm.nr_hugepages_mempolicy = 1024
   vm.nr_overcommit_hugepages = 0
   ```

5. Enable huge pages in /etc/default/qemu-kvm or create file if it does not exist with the following option:

   ```
   KVM_HUGEPAGES=1
   ```

6. Reserve 1024 huge pages of 2MB size.

   ```
   sudo sh -c "echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages"
   ```

7. Verify that the pages have been reserved:

```
grep Huge /proc/meminfo
```

Example of system response:

```
AnonHugePages:   1138688 kB
HugePages_Total:    1024     # here we see that we have 1024 hugepages
HugePages_Free:     1024     # all pages are free
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB  # size of each page equals 2 MB
```

8. Add the mount point for the hugetlbfs file system to /etc/libvirt/qemu.conf:

```
hugetlbfs_mount = "/run/hugepages/kvm"  # this is default mount point for huge pages
```

9. Restart the qemu-kvm service:

```
sudo service qemu-kvm restart
```

10. Restart the libvirt-bin service:

```
sudo service libvirt-bin restart
```

11. Restart the nova-compute service.

```
sudo service nova-compute restart
```

Seealso

- PPA for Ubuntu packages
- Ubuntu Trusty cloud image

Boot a virtual machine with Huge Pages

This scenario demonstrates booting VM with Huge Pages.

To boot a virtual machine with Huge Pages:

1. Create a new flavor or use an existing one to use with Huge Pages. If you need to create a new image flavor, run:

```
. openrc admin admin        # get admin rights
nova flavor-create huge 999 1024 4 1
```

2. Add the size of huge pages to the image flavor.

```
nova flavor-key huge set hw:mem_page_size=2048
nova flavor-show huge
```

   Example of system response

```
+----------------------------+------------------------------+
| Property                   | Value                        |
+----------------------------+------------------------------+
| OS-FLV-DISABLED:disabled   | False                        |
| OS-FLV-EXT-DATA:ephemeral  | 0                            |
| disk                       | 4                            |
| extra_specs                | {"hw:mem_page_size": "2048"} |
| id                         | 7                            |
| name                       | huge                         |
| os-flavor-access:is_public | True                         |
| ram                        | 1024                         |
| rxtx_factor                | 1.0                          |
| swap                       |                              |
| vcpus                      | 1                            |
+----------------------------+------------------------------+
```

3. Create a new image or use an existing image. You need an Ubuntu image and the default Cirros image. If you need to create a new Ubuntu image:

```
glance --os-image-api-version 1 image-create --name ubuntu \
        --location https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.im
        --disk-format qcow2 --container-format bare
```

4. Boot a new instance using the created flavor.

```
nova boot --flavor huge --image ubuntu inst1
```

5. After booting the VM, verify that the VM uses 512 huge pages.

```
grep Huge /proc/meminfo
```

Example of system response

```
AnonHugePages:   1138688 kB
HugePages_Total:    1024
HugePages_Free:      512
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

Glossary

NFV Glossary

Term	Definition
ARI	Alternative Routing-ID Interpretation
DPDK	Data Plane Development Kit
DMA	Direct Memory Access
Huge Pages	The Linux kernel mechanism that enables multiple page size capabilities of modern hardware architectures
IOAPIC	Input/Output Advanced Programmable Interrupt Controller
IOMMU	Input/Output Memory Management Unit
KVM	Kernel-based Virtual Machine
MMU	Memory Management Unit
MSI	Message Signaled Interrupt
NUMA	Non-Uniform Memory Access
OVS	Open vSwitch
PCI	Peripheral Component Interconnect
PF	Physical Function
PMD	Poll-mode driver
PPA	Personal package archive
QEMU	Quick Emulator
SR-IOV	Single Root Input/Output Virtualization
VF	Virtual Function
VHOST _USER	DPDK VHOST_USER implementation
VIF	VM’s Virtual Interface on the host sid
VM	Virtual Machine