CONFIGURATION GUIDE

Enabling Enhanced Platform Awareness for Superior Packet Processing in OpenStack*

Intel Corporation
Datacenter Network
Solutions Group

Authors

Shivapriya Hiremath
Solution Software Engineer,
Intel Corporation

Vaidyanathan Krishnamoorthy
Solution Software Engineer,
Intel Corporation

Andrew Duignan
Solution Software Engineer,
Intel Corporation

James Chapman
Solution Software Engineer,
Intel Corporation

Nikita Agarwal
Solution Software Engineer,
Intel Corporation

Tarek Radi
Lead Technical Program Manager,
Intel Corporation

1.0 Introduction

Network functions virtualization (NFV) emerged in response to the problems that network operators face when developing and maintaining their infrastructures built with traditional hardware appliances. The growing demands for new services, constantly increasing network traffic, and the need for rapid service delivery are just some of the challenges that industry cannot overcome with the traditional network architectures. Moreover, the technological advancements cause the hardware life cycle to shorten. This leads to long-lasting procure-design-deploy cycles that require unique human skills.

NFV is a network architecture concept designed by a group of telecommunications industry representatives of the European Telecommunications Standards Institute (ETSI). NFV primarily aims to leverage the existing technologies to virtualize entire classes of network nodes that in turn may be connected to form compound infrastructures. Designed with modularity, reliability, and scalability in mind, such infrastructures are capable of running complex services.

The architecture of NFV consists of three blocks: virtualized network functions (VNFs), NFV infrastructure (NFVI), and NFV management and orchestration (MANO). One of the MANO components of the ETSI NFV model, virtualized infrastructure manager (VIM), is responsible for managing and orchestrating orchestrating the NFVI.

The NFVI consists of physical compute, storage, and networking components that via its virtualization layer provides pool of virtual resources to run the VNFs. Intel is uniquely positioned to speed up the NFV development. In the Open Platform for NFV* (OPNFV*), an open-source implementation of the NFV specification, OpenStack* has been selected to serve as the VIM.

This document is intended for network administrators and architects planning to implement or optimize virtualized infrastructures, and should be treated as a configuration guide complementary to the OpenStack Enhanced Platform Awareness white paper available at https://networkbuilders.intel.com/docs/ice-house-openstack-enhanced-platform-awareness.pdf. It explains how to configure OpenStack to optimally use the capabilities of the underlying Intel® architecture-based servers, and how to deliver improved performance and enhanced predictability for virtualized applications.

Table of Contents

1.1 OpenStack

OpenStack is a leading open-source software suite for creating private and public clouds with code first released in 2010 under the Apache* 2.0 license. Since then, it has grown in popularity with an active community of users and contributors.

OpenStack is used to manage pools of NFVI resources that typically are based on standard, high-volume servers (SHVS) and can be partitioned and provisioned on demand with a command line interface (CLI), RESTful API, or a web interface.

The OpenStack Compute* service, Nova, is responsible for managing all compute infrastructure in an OpenStack-managed cloud. OpenStack supports multiple hypervisor drivers, including QEMU*/KVM* (by means of libvirt*), Xen*, and VMware vSphere* Hypervisor (VMware ESXi*). OpenStack Compute contains the scheduling functionality that is used to select which compute host runs a particular workload. OpenStack Compute filters all available platforms to a suitable subset based on the input requirements, and then selects a platform from that subset based on a weighting algorithm.

The OpenStack Networking* service (Neutron), is designed as a scalable service that offers a variety of plug-in solutions to help service providers with network management. OpenStack Networking also offers a self-service interface to customers so they can create their own networks based on available network models such as a flat network, virtual LAN (VLAN), or overlay networks such as virtual extensible LAN (VxLAN) and Network Virtualization using Generic Routing Encapsulation (NVGRE).

1.2 Enhanced Platform Awareness

From the NFV perspective, OpenStack does not require any fore knowledge of the NFV applications or their functions; however, OpenStack provides an advanced selection of tuning capabilities that enable service providers to deploy NFV solutions with the necessary performance and efficiency characteristics.

Enhanced Platform Awareness (EPA) is a set of contributions from Intel Corporation and others to OpenStack. EPA features provide OpenStack a better view of the underlying hardware and enable OpenStack to filter platforms with specific capabilities that match the workload requirements, prior to launching a virtual machine (VM). For example, EPA can automatically launch a cryptographic workload on a platform with a hardware-based cryptographic accelerator.

For workloads requiring particular CPU or I/O capabilities, EPA helps OpenStack VMs to run on the optimal platforms. EPA can benefit VM performance and operation, such as for software-defined networking (SDN) and NFV. EPA also enables cloud service providers to offer premium, revenue-generating services based on specific hardware features.

The list of features supported by EPA is presented in Appendix A: Summary of EPA Features, while Figure 1 shows which OpenStack release first supported the particular EPA feature.

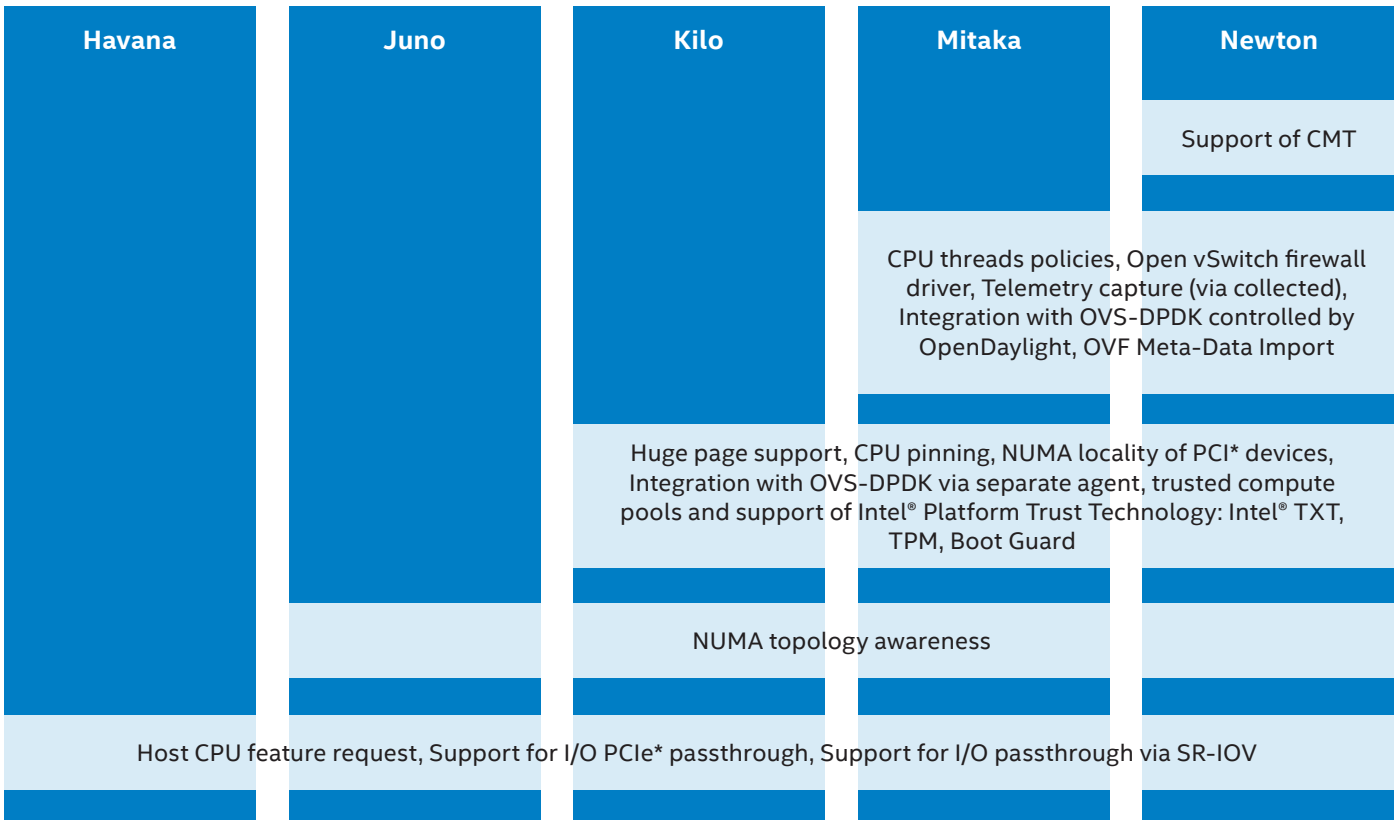


Figure 1. The history of EPA features in OpenStack.

2.0 Configuration and Enablement

This section describes the instructions needed to enable various EPA features available in OpenStack. Make sure you are able to run the commands in the controller's CLI. Depending on the OpenStack variant installed, you may have to source the appropriate OpenStack RC credential file. For more information, visit http://docs.openstack.org/user-guide/common/cli-set-environment-variables-using-openstack-rc.html.

2.1 Host CPU Feature Request

In the OpenStack Icehouse release, a change was made to the OpenStack Compute libvirt driver in order to expose all of the CPU instruction set extensions to the nova-scheduler. This correlated with an associated change in the libvirt to make this data available by means of libvirt API. These changes enabled the creation of OpenStack Nova flavors that contain specific feature requests by adding these to the flavor as the `extra_specs` parameter.

```
# nova flavor-key <flavorname> set
capabilities:cpu_info:features=<feature
name, for example aes>
```

During scheduling, the `compute_capabilities_filter` in the OpenStack Compute service compares the requirements on the host CPU as specified by the flavor's `extra_specs` parameter with a database of hosts and their respective CPU features. To enable the CPU feature request, configure the libvirt to expose the host CPU features to the guest by setting the following parameter in the `/etc/nova/nova.conf` file.

```
[libvirt]
cpu_mode=host-model or host-passthrough
or custom or none
```

- The host-model causes libvirt to identify the named CPU model that most closely matches the host CPU from the libvirt's list of standard CPU names defined in `the/usr/share/libvirt/cpu_map.xml` file and then to request additional CPU flags to complete the match. This option should keep the VM's functionality and best possible performance that still maintains acceptable reliability and compatibility if the guest is migrated to another host with slightly different CPUs.
- The `host-passthrough` causes `libvirt` to instruct the kernel-based virtual machine (KVM) to pass through the host CPU with no modifications. Comparing to the `host-model`, every detail of the host CPU is matched instead of matching just feature flags. This gives absolutely the best performance and can be important to applications that check low-level CPU details. The guest can only be migrated to the host with an exactly matching CPU.
- The `custom` option allows the use of a named CPU model.
- The `none` option provides the hypervisor with the default configuration.

2.2 Support for I/O PCIe* Passthrough

OpenStack includes the support for full device passthrough and Single Root I/O Virtualization (SR-IOV) for non-networking devices and networking devices not managed by OpenStack Neutron. These features enable the allocation of physical functions (PFs) or virtual functions (VFs) to the VM from Peripheral Component Interconnect Express* (PCIe*) devices.

The following steps show how to configure the Peripheral Component Interconnect* (PCI*) passthrough support in the OpenStack by updating the `nova.conf` files on compute and controller nodes.

On the compute node:

1. Configure `pci_passthrough_whitelist` with the details of PCI devices available to VMs in `nova.conf`, for example,

```
pci_passthrough_whitelist =[{ "vendor_
id":"8086","product_id":"1520", "device_
type":"NIC"}]
```

This defines that the platform's PCI devices with the `vendor_id` as `0x8086` and the `product_id` as `0x1520` will be assignable to the instances.

On the controller node:

2. Configure an alias for the PCI passthrough device in `nova.conf`, for example,

```
pci_alias={"vendor_id":"8086", "product_
id":"1520", "name":"a1", "device_
type":"NIC"}
```

This defines the `pci_alias` named `a1` for PCI devices with the `vendor_id` as `0x8086` and the `product_id` as `0x1520`.

3. Enable PCI devices filter for the scheduler in `nova.conf`, for example,

```
scheduler_driver=nova.scheduler.filter_
scheduler.FilterScheduler
scheduler_available_filters=nova.
scheduler.filters.all_filters
scheduler_available_filters=nova.
scheduler.filters.pci_passthrough_filter.
PciPassthroughFilter
scheduler_default_filters=RamFilter,Comput
eFilter,AvailabilityZoneFilter,ComputeCapa
bilitiesFilter,ImagePropertiesFilter,PciPa
ssthroughFilter
```

The steps below show how to use PCI passthrough in VMs.

1. Create and configure a flavor that requires PCI devices, for example,

```
# nova flavor-key m1.large set "pci_
passthrough:alias"="a1:2"
```

This updates a flavor that requires two PCI devices, each with the `vendor_id` as `0x8086` and the `product_id` as `0x1520`.

2. Create a key pair in OpenStack.

```
# nova keypair-add sshkey
```

3. Create a VM, for example,

```
# nova boot --image new1 --key_name
sshkey --flavor m1.large 123
```

This command creates a VM with the PCI device attached. The VM image contains the driver for the assigned PCI devices, and `sshkey` is the name of the key pair created in Step 2.

Go to https://wiki.openstack.org/wiki/Pci_passthrough for more information on PCI passthrough.

2.3 Support for I/O Passthrough via SR-IOV

This section describes the steps required to use the SR-IOV extensions for NICs. To enable I/O passthrough, prepare the system with the following steps.

1. To check whether Input-Output Memory Management Unit (IOMMU) is supported, run the following command. The output should show IOMMU entries.

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

Note: IOMMU can be enabled/disabled through a BIOS setting, under **Advanced→Processor**.

2. Enable Intel® Virtualization Technology for Directed I/O in BIOS.

3. Enable IOMMU on the host kernel by adding the following parameter to the Grand Unified Bootloader (GRUB) boot parameters in the `/etc/default/grub` file.

```
GRUB_CMDLINE_LINUX="intel_iommu=on
iommu=pt"
```

4. Update GRUB with the following command and then reboot the server.

```
# grub2-mkconfig –o /boot/grub2/grub.cfg
```

5. On the host, ensure that device drivers for the NICs are configured to enable the NIC VFs.

6. Install the necessary packages: `yajl-devel, device-mapper-devel, libpciaccess-devel, libnl-devel, dbus-devel, numactl-devel, python-devel`, including `libvirt` and `libvirt-python`.

7. Modify the `/etc/libvirt/qemu.conf` file by adding `/dev/vfio/vfio to the cgroup_device_acl` list, for example:

```
cgroup_device_acl = ["/dev/null", "/dev/
full", "/dev/zero", "/dev/random", "/dev/
urandom", "/dev/ptmx", "/dev/kvm", "/dev/
kqemu", "/dev/rtc", "/dev/hpet", "/dev/net/
tun", "/dev/vfio/vfio"]
```

8. Enable the SR-IOV VF for a network interface. The following example enables two VFs for the interface `p1p1`.

```
# echo 2 > /sys/class/net/p1p1/device/
sriov_numvfs
```

To check that VFs are enabled, execute the following command.

```
# lspci -nn | grep 82599
```

The output of this command should show one PF and two VFs instantiated. The user can use either the PCI device's VF or PF for SR-IOV. In this case we use a PF. This means that all the associated VFs will be made available to VMs.

9. A recommended practice is to create a host aggregate for the platforms that have additional tuning for NFV.

```
# nova aggregate-create nfv-aggregate
# nova aggregate-set-metadata nfv-
aggregate nfv=true
```

10. Update the flavor to enable the `nfv-aggregate` metadata (`nfv=true`).

```
# nova flavor-key <flavor-name> set
aggregate_instance_extra_specs:nfv=true
```

11. In the OpenStack Compute configuration file, `nova.conf`, the white list needs to be configured to enable the VF identifiers to be shared with the `nova-scheduler`, and the `PciPassthroughFilter` needs to be enabled.

Note: The PCI alias is not required when managing the SR-IOV networking device with OpenStack Networking.

```
pci_passthrough_whitelist={"address":"0000
:08:00.0","physical_network":"physnetNFV"}
scheduler_default_filters = <default
list>,PciPassthroughFilter
```

OpenStack Networking requires that the SR-IOV modular layer 2 (ML2) mechanism driver is used and the VF vendor and device IDs are set up. To enable the usage of SR-IOV ML2 mechanism driver, configure the `/etc/neutron/plugins/ml2/ml2_conf.ini` file, for example:

```
[ml2]
tenant_network_types = vlan
type_drivers = vlan
mechanism_drivers =
openvswitch,sriovnicswitch
[ml2_type_vlan]
network_vlan_ranges = physnetNFV:50:100
```

To set the VF vendor and device ID, configure the `/etc/neutron/plugins/ml2/ml2_conf_sriov.ini` file, for example:

```
[ml2_sriov]
supported_pci_vendor_devs = 8086:10fb
agent_required = False
[sriov_nic]
physical_device_mappings =
physnetNFV:eth1
```

12. To apply the configuration, restart the `neutron-server`.

```
# systemctl restart neutron-server
```

13. With the OpenStack Networking application programming interface (API), the tenant must create a port with a virtual NIC type (`vnic-type`) as `direct`. This means that the VF will be allocated directly to the VM.

```
# neutron net-create –provider:physical_
network=physnetNFV -provider:network_
type=vlan
```

14. Create a network called NFV-network.

```
# neutron subnet-create NFV-network
<CIDR> –name <Subnet_Name> –
allocationpool=< start_ip>, end=<end_ip>
# neutron port-create NFVnetwork
--binding:vnic-type direct
```

15. Create a VM. The port ID that is returned from the neutron port-create command must be added to the nova boot command.

Note: During the boot process, OpenStack Compute will check the validity of this port ID with OpenStack Networking.

```
# nova boot --flavor <flavorname> --image
<image> --nic port-id=<from port-create
command> <vm name>
```

Go to https://wiki.openstack.org/wiki/SR-IOV-Passthrough-For-Networking-Mitaka-Ethernet for more information on SR-IOV passthrough for OpenStack Networking.

2.4 NUMA Topology Awareness

Awareness of NUMA topology in the platform was added in the OpenStack Juno release with the 'Virt driver guest NUMA node placement and topology' extension. This feature allows the tenant to specify its desired guest NUMA configuration. The nova-scheduler was extended with the NUMATopologyFilter to help match guest NUMA topology requests with the available NUMA topologies of the hosts.

```
scheduler_default_filters = <default
list>, NUMATopologyFilter
```

Tenants can specify their request by means of an OpenStack Compute's flavor-based mechanism. An example of such a command is:

```
# nova flavor-key <flavor-name> set
hw:numa_mempolicy=strict hw:numa_
nodes=2 hw:numa_cpus.0=0,1,2,3 hw:numa_
cpus.1=4,5,6,7 hw:numa_mem.0=1024 hw:numa_
mem.1=1024
```

Tenants also have the option to specify their guest NUMA topology request by means of an image-property-based mechanism. An example of such a command is:

```
# glance image-update image_id –property
hw_numa_mempolicy=strict –property
hw_numa_cpus.0=0,1,2,3 –property hw_
numa_cpus.1=4,5,6,7 –property hw_numa_
mem.0=1024 –property hw_numa_mem.1=1024
```

These commands result in OpenStack configuring the guest virtual CPUs 0, 1, 2, and 3 to be mapped to socket 0 (also known as cell 0 in libvirt terminology) and virtual CPUs 4, 5, 6, and 7 to be mapped to socket 1.

2.5 NUMA Locality of PCI Devices

The OpenStack Kilo release enables I/O-aware NUMA scheduling. To use this capability with PCIe passthrough, the flavor must be updated to request the particular PCIe device.

```
# nova flavor-key <flavorname> set "pci_
passthrough:alias"="niantic:1"
```

In this example, the alias is the same as was configured in section 2.2 Support for I/O PCIe* Passthrough, and the number 1 represents the number of VFs that must be allocated. PciPassthroughFilter finds hosts with the requested PCIe devices. If the I/O device is a network device and the vnic-type set with the neutron port-create command is direct or macvtap, the physical_network setting that was used with the neutron net-create command is also taken into account by the scheduler to identify the suitable set of hosts. Next, the NUMATopologyFilter filter selects the NUMA node and checks whether the NIC is locally attached.

2.6 CPU Pinning

By default virtual CPUs (vCPUs) are not pinned to physical CPUs (pCPUs), meaning that application can utilize any host CPU core that has been assigned to it by the virt driver. This approach ensures the optimal performance of the system that is heavily loaded at the possible expense of the performance of individual instances.

Applications that require real-time or near real-time behavior must avoid resource contention that can be mitigated by dedicating a pool of physical cores to them. This functionality is known as CPU pinning, or more precisely, pinning guest's vCPUs to pCPUs, and ensures that only the selected application will be executed on this pool of cores, and no other processes will be scheduled on that pool.

OpenStack Kilo release adds a capability to pin guest vCPUs to the specific pCPUs on the host that will further allow for execution of entire VMs on the dedicated pCPUs. In OpenStack, CPU pinning can be enabled by specifying the hw:cpu_policy property of the extra_specs parameter in the OpenStack Compute's flavor. The default CPU policy is shared denoting that guest vCPUs can freely float across the physical cores on the host. A dedicated CPU policy enables pinning of guest vCPUs to pCPUs, and provides the option to select one of the selected vCPU thread policies, described in the next section.

```
hw:cpu_policy=shared|dedicated
```

When using dedicated CPU policy, isolate CPUs from the host operating system (OS) on the platform to prevent the guest and the host from contending for resources on the same cores. To prevent the host from using specific cores, use the isolcpus setting in GRUB command line. To avoid contention on execution units from shared CPU siblings, such as between pCPU 0 and pCPU 4, isolate sibling CPUs from the host. Based on the CPU numbering, if it is necessary to dedicate an entire execution unit from each socket to the host, then a possible isolation strategy would be to allocate pCPUs 0, 4, 8, and 12 to the host, and to isolate the other pCPUs.

Edit the /etc/default/grub file to isolate specific CPU cores from OS, for example:

```
GRUB_CMDLINE_LINUX="isolcpus= 1, 2, 3, 5,
6, 7, 9, 10, 11, 13, 14, 15"
```

2.7 CPU Threads Policies

This functionality is complementary to OpenStack's CPU pinning and has been introduced to OpenStack Mitaka release with four CPU thread policies defined. Each CPU thread policy configures the mapping of guest vCPUs to physical CPUs (pCPUs) on the host in the context of a system enabled with simultaneous multi-threading (SMT).

An SMT-enabled, Intel architecture-based platform has two CPU hardware threads that can execute simultaneously on each core execution unit. To the kernel, this appears to be twice as many cores in the system as are actually available.

```
hw:cpu_threads_policy=avoid|separate|isol
ate|prefer
```

The `hw:cpu_threads_policy` controls how the scheduler (virt driver) is placing guests on CPU threads. It only applies if the scheduler policy is set to `dedicated`. The following policies have been defined for OpenStack Mitaka release:

- `avoid`: the scheduler will not place the guest on a host which has hyper-threaded cores enabled with Intel® Hyper-Threading Technology.
- `separate`: if the host has hyper-threaded cores, each vCPU will be placed on a different core; thus, no two vCPUs will be placed on thread siblings.
- `isolate`: if the host has hyper-threaded cores, each vCPU will be placed on a different core and no vCPUs from other guests will be placed on the same core; thus, one thread sibling is guaranteed to always be unused.
- `prefer`: if the host has hyper-threaded cores, vCPU will be placed on the thread sibling.

Go to https://networkbuilders.intel.com/docs/CPU_Pinning_With_Openstack_nova.pdf for more information on CPU pinning and CPU thread policies.

2.8 Huge Page Support

The OpenStack Kilo release supports huge page capabilities on hosts. To leverage huge pages, the host OS must be configured to define the huge page size and the number of huge pages to be created. As shown in the following example, the default huge page size setting is 2 MB, with eight 1 GB pages allocated.

1. To set huge pages, edit the `/etc/default/grub` file.

```
GRUB_CMDLINE_LINUX="default_
hugepagesz=2MB hugepagesz=1G hugepages=8"
```

2. Update GRUB.

```
# grub2-mkconfig –o /boot/grub2/grub.cfg
```

3. In addition, libvirt on the host must be configured to allow the use of huge pages. In the `/etc/libvirt/qemu.conf` file, add the hugetlbfs mount point, for example, `/mnt/huge`, to the `cgroup_device_acl` list.

```
cgroup_device_acl = [
      "/dev/null", "/dev/full", "/dev/zero",
      "/dev/random", "/dev/urandom",
      "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
      "/dev/rtc", "/dev/hpet","/dev/net/tun",
      "/mnt/huge"
]
```

4. Restart libvirt to apply the changes. Mount `hugetlbfs` and allocate space with the following command.

```
# mount –t hugetlbfs –o
mode=0777,pagesize=1G,size=16G none /mnt/
huge
```

The `nova-scheduler` has been updated to track and allocate these huge pages to guest operating systems. The flavor definition should contain the page size that is required for the VM. The following example shows a request for 1 GB page size.

```
# nova flavor-key <flavor name> set
hw:mem_page_size=1048576
```

2.9 Trusted Compute Pools

To improve security, cloud subscribers may want their workloads to be executed on verified compute platforms. In OpenStack, it is possible to designate groups of trusted hosts to form pools of trusted computes.

The trusted hosts are supported with hardware-based security features, such as Intel® Trusted Execution Technology (Intel® TXT), and 3rd party attestation servers. Attestation servers are offered, for example, by Open Cloud Integrity Technology (Open CIT) and perform host verification activities.

With the use of trusted compute pools, subscribers are ensured that their workloads will run on trusted compute nodes, and OpenStack cloud providers will offer secure stacks that will allow only the verified software to be run.

Perform the following steps to configure the OpenStack Compute to be used in trusted compute pools.

1. Enable scheduling support for trusted compute pools and specify the connection details for the attestation service by adding the following lines to the `/etc/nova/nova.conf` file:

```
[DEFAULT]
compute_scheduler_driver=nova.scheduler.
filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler
.filters.all_filters
scheduler_default_filters=Availability
ZoneFilter,RamFilter,
ComputeFilter,TrustedFilter


[trusted_computing]
attestation_server = <IP of the
attestation service>
attestation_port = <HTTPS port for the
attestation service>
attestation_server_ca_file = <certificate
file used to verify the identity of the
attestation server>
attestation_api_url = <attestation
service's URL path>
attestation_auth_blob = i-am-openstack
```

2. Restart the `nova-compute` and `nova-scheduler` services.

Perform the following steps to specify the trusted flavor.

1. Set the requested flavor as trusted.

```
# nova flavor-key <flavor_name> set
trust:trusted_host=trusted
```

2. Request the instance to be executed on a trusted host by specifying a trusted flavor when booting the instance.

```
# nova boot --flavor <flavor_name> --key-
name <keypair_name> --image <image_id>
<new_instance_name>
```

More information on trusted compute pools can be found at http://docs.openstack.org/admin-guide/compute-security.html and https://software.intel.com/sites/default/files/managed/2f/7f/Config_Guide_for_Trusted_Compute_Pools_in_RHEL_OpenStack_Platform.pdf.

2.10 Open vSwitch* Firewall Driver

The Open vSwitch* (OvS*) firewall driver is fully integrated with the Data Plane Development Kit-accelerated OvS (OVS-DPDK). Rather than leveraging the Linux bridge and iptables, it natively implements security groups as flows in the OvS. Thus, it creates a pure OvS model that is not dependent on functionality from the underlying platform. This firewall driver uses the same public API to talk to the OpenStack Networking agent as the existing Linux bridge firewall implementation.

To use firewall driver in combination with OVS-DPDK, download the networking-ovs-dpdk project.

```
# git clone https://github.com/stackforge/
networking-ovs-dpdk.git
```

Also, you may want to install this project as a `pip` package.

```
# pip install networking-ovs-dpdk
```

To enable the firewall driver, add the following section to your `devstack/local.conf` file. Alternatively, if DevStack* is not used to deploy your OpenStack cloud, modify the `/etc/neutron/plugins/ml2/ml2_conf.ini` file accordingly.

```
[[post-config|/etc/neutron/plugins/ml2/
ml2_conf.ini]]
[securitygroup]
#firewall_driver = neutron.agent.linux.
iptables_firewall
firewall_driver = networking_ovs_dpdk.
agent.ovs_dpdk_firewall.OVSFirewallDriver
```

More information on the OvS firewall driver can be found at https://software.intel.com/en-us/articles/implementing-an-openstack-security-group-firewall-driver-using-ovs-learn-actions and https://github.com/openstack/neutron/blob/master/doc/source/devref/openvswitch_firewall.rst.

2.11 Support for OVS-DPDK Controlled by OpenStack Networking*

OpenStack's networking-as-a-service project, OpenStack Networking (Neutron), is able to use the functionality of a distributed, virtual, multilayer switch by using `networking-ovs-dpdk`. It represents a collection of deployment scripts that enable OVS-DPDK-based deployments.

The project started by providing a custom, out-of-tree implementation of a Data Plane Development Kit (DPDK)-enabled version of the OvS agent and ML2 driver along with DevStack-based deployment scripts. These applications allow users to attach DPDK-backed vHost-user ports to their instances, which can provide up to a 10x performance improvement over the standard OvS ports.

More details about OVS-DPDK and how it can be enabled in OpenStack can be found at https://01.org/openstack/blogs/stephenfin/2016/enabling-ovs-dpdk-openstack.

2.12 Support for OVS-DPDK controlled by OpenDaylight*

OpenDaylight* is software-defined network (SDN) controller that lets the user programmably manage OpenFlow*-capable switches. OpenStack can use OpenDaylight as its network management provider through the ML2 north-bound plug-in. OpenDaylight manages the network flows for the OpenStack compute nodes via the Open vSwitch database (OVSDB) and the OpenFlow south-bound plug-ins.

More information on the setup of OpenStack with OVS-DPDK controlled by OpenDaylight can be found at https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/getstarted/devstack/ubuntu.rst.

2.13 Telemetry Capture (via collectd)

collectd is a highly-scalable, system statistics collection service, written in C. It has many metrics already available, such as CPU utilization, FS-Cache, IRQ handling, and network interface and routing statistics. It has a plug-in-based architecture, which means that only chosen measurements may be enabled, making the process very lightweight to run.

OpenStack is written in Python, and collectd provides language bindings through its Python plug-in. A collectd plug-in was developed in Python to interact with the OpenStack Telemetry* service (Ceilometer).

This plug-in makes the metrics from collectd available to the OpenStack Telemetry service. This means that all the previously available metrics from collectd can be made available to the OpenStack Telemetry, and plug-ins such as `interface`, `cpu`, and `cpufreq` can be used to get vital statistics about network load and CPU usage. This data can then be used for scheduling, for example, or as a diagnostic tool, for identifying performance bottlenecks.

In addition, these statistics can be used by any agent in the OpenStack cloud, and some logic or intelligence can be applied to make more informed decisions for automatic provisioning of resources. For example, additional statistics could be used to identify potential faults in a system and evacuate all workloads to ensure continuous service.

More information on how to install and use collectd can be found at: https://collectd.org/wiki/index.php/First_steps.

3.0 Intel® Technologies for Enhanced Platform Awareness

Following section presents an overview of technologies and platform capabilities supported by OpenStack through the EPA features.

3.1 Intel® Hyper-Threading Technology

The Intel® Xeon® processors that are used in the servers support Intel® Hyper-Threading Technology (Intel® HT Technology). Intel HT Technology enables CPU resources to be used more efficiently, because multiple threads can be run on each core.

Intel HT Technology is enabled in the BIOS settings and is supported by most Linux* flavors. As a performance feature, Intel HT Technology also increases CPU throughput, improving overall performance on threaded software, and providing headroom for future business growth and new solution capabilities. Refer to sections 2.6 and 2.7 for more details on setting the appropriate thread policy in the OpenStack.

More information on the Intel® Hyper-Threading Technology can be found at http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html.

3.2 Intel® Resource Director Technology: Cache Monitoring Technology and Cache Allocation Technology

The Cache Monitoring Technology (CMT) feature allows the OS, hypervisor, or virtual machine monitor (VMM) to determine the cache usage of applications running on the platform. CMT can be used to do the following:

- Detect if the platform supports CMT monitoring capabilities via CPUID
- Have the OS or VMM assign the resource monitoring ID (RMID) for each application or VM scheduled to run on a core
- Monitor cache occupancy and memory bandwidth on a per-RMID basis
- Allow the OS or VMM to read shared last-level cache (LLC) occupancy and memory bandwidth for a given RMID at any time

The Cache Allocation Technology (CAT) feature allows an OS, hypervisor, or VMM to control the allocation of a CPU's shared LLC. Once CAT is configured, the processor allows access to portions of the cache according to the established class of service (CLOS). The processor obeys the CLOS rules when it runs an application thread or application process.

The following stock keeping units (SKUs) of Intel Xeon processors support both CAT and CMT:

- Intel® Xeon® processor E5-2658 v3
- Intel® Xeon® processor E5-2658A v3
- Intel® Xeon® processor E5-2648L v3
- Intel® Xeon® processor E5-2628L v3
- Intel® Xeon® processor E5-2618L v3
- Intel® Xeon® processor E5-2608L v3
- All SKUs of the Intel® Xeon® processor D product family
- All SKUs of the Intel® Xeon® processor E5-2600 v4 product family

Go to https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html to read more on CAT and CMT as the subset of Intel® Resource Director Technology.

3.3 Intel® Advanced Encryption Standard New Instructions

Intel® Advanced Encryption Standard New Instructions is an instruction set that enables improved speed encryption/decryption of workloads using the Advanced Encryption Standard (AES). This feature was added to 4th Generation Intel® Core™ processor family and Intel® Xeon® processor family that base on the Intel® microarchitecture code name Westmere.

Go to https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni to read more on Intel AES-NI.

3.4 Intel® Advanced Vector Extensions

Intel® Advanced Vector Extensions (Intel® AVX) is a set of single instruction, multiple data (SIMD) instructions introduced in the Intel® microarchitecture code name Sandy Bridge.

Intel® Advanced Vector Extensions 2 (Intel® AVX2) expands most vector interger Streaming SIMD Extensions (SSE) and Intel AVX instructions to 256 bits. Intel® AVX2 support was first added to 4th generation Intel® Core™ processor family for client systems and Intel® Xeon® v3 processor family for server systems (formerly code-named Haswell).

Go to https://software.intel.com/en-us/articles/how-intel-avx2-improves-performance-on-server-applications to read more on Intel AVX and Intel AVX2.

3.5 Intel® Streaming SIMD Extensions 4.2

Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) is an extension to the Intel® SSE instructions set that enables SIMD parallelization. This feature was added to the Intel® Xeon® processor family that base on the Intel® microarchitecture code name Nehalem.

Go to http://www.intel.com/content/www/us/en/support/processors/000005779.html to read more on Intel SSE4.2.

3.6 RDRAND

RDRAND is an instruction for returning random numbers. This was introduced to 3rd generation Intel® Core™ processor family for client systems and Intel® Xeon® processor family for server systems that base on the Intel® microarchitecture code name Ivy Bridge.

Go to https://software.intel.com/en-us/blogs/2011/06/22/find-out-about-intels-new-rdrand-instruction to read more on RDRAND.

3.7 Intel® Trusted Execution Technology

Intel® Trusted Execution Technology (Intel® TXT) is a combination of hardware and software, aimed at helping secure the execution of sensitive workloads. In contrast to solutions that protect the OS, Intel TXT builds a chain of trust from the system firmware, all the way to the server or hypervisor to help prevent attacks on the system firmware or BIOS, master boot record, boot loader, OS, and hypervisor.

OpenStack Grizzly and newer versions provide a `TrustedFilter` to scheduler's filter that uses Intel TXT to schedule workloads requiring trusted execution only to trusted compute resources.

Go to http://docs.openstack.org/admin-guide/compute-security.html to read more on how to take advantage of Intel TXT for creating a trusted computed pool and make use of it in OpenStack.

3.8 Intel® QuickAssist Technology

Intel QuickAssist Technology provides security and compression acceleration capabilities used to improve performance and efficiency across the data center. Server, networking, big data, and storage applications use the Intel QuickAssist Technology to offload servers from handling compute-intensive operations, such as:

- Symmetric cryptography functions including cipher operations and authentication operations
- Public key functions including RSA, Diffie-Hellman, and elliptic curve cryptography
- Compression and decompression functions including Deflate algorithm

Ultimately, the Intel QuickAssist Technology helps users to ensure applications are fast, secure, and available, and enables the demands of ever-increasing amounts of data, especially data with the need for encryption and compression, to be met.

The instructions on enabling the Intel QuickAssist Technology in the OpenStack are similar to the steps listed in section 2.2 Support for I/O PCIe* Passthrough.

More information on Intel QuickAssist Technology can be found at http://www.intel.com/content/www/us/en/embedded/technology/quickassist/overview.html.

Appendix A: Summary of EPA Features

EPA FEATURE	DESCRIPTION	RELATED PLATFORM CAPABILITIES	SUPPORTED BY NFVI	VNF / VNF MANAGER SUPPORT REQUIREMENT
Host CPU Feature Request	Exposes all of the CPU instruction set extensions to the nova-scheduler OpenStack service. Exposure of these CPU capabilities to the guest OS improves VNF performance in specific areas, for example, encryption/decryption.	Intel® AES-NI Intel® AVX2 Intel® SSE4.2 RDRAND	Yes	Based on the VNF requirement, VM must be created with the host CPU feature request.
Support for I/O PCIe Passthrough	Provides I/O functionality to the guest OS and improves I/O performance bypassing the host OS. Value: Supports full device passthrough for non-networking devices and networking devices not managed by OpenStack Networking*.	Enhanced platform PCI/PCIe devices such as NICs, cryptography accelerators based on Intel® QuickAssist Technology, and so on.	Yes	The guest OS must include the PCI PF driver for the I/O passthrough device. Additionally, the VNF descriptor must be updated for the VNF manager.
Support for I/O Passthrough via SR-IOV	Passthrough of virtualized hardware functions to the guest OS, and thus improving performance and scalability of I/O. Value: Allows an I/O device to be shared by multiple VMs without losing runtime performance.	SR-IOV	Yes	The guest OS must include the PCI device VF driver for the I/O passthrough device. Additionally, the VNF descriptor must be updated for the VNF manager.
NUMA Topology Awareness NUMA Locality of PCI Devices	Allows the use of particular CPU cores, memory, and I/O devices assigned to a specific NUMA node. Value: Increases the effective utilization of compute resources and decreases latency by avoiding cross-node memory accesses by the guests.	NUMA architecture of the host	Yes	VM must be created with the feature request based on the VNF requirement. VNF exposes the requirement through the descriptor.
CPU Pinning	Allows the vCPUs used by a guest to be tied to the pCPUs on the host. Value: Improves the performance of the guest by preventing resource contention with other guest instances and host processes.	Intel® HT Technology	Yes	VM must be created with the specific CPU pinned. VNF exposes the requirement through the descriptor.
CPU Threads Policies	Controls the allocation of CPU thread siblings to vCPUs to ensure the best performance of VM. Value: Improves VM performance by enabling service providers to decide how guest vCPUs will utilize the physical CPU cores.	Intel® HT Technology	Yes	VM must be created with the specific CPU pinned. VNF exposes the requirement through the descriptor.
Huge Page Support	Supports memory pages greater than the default size (usually 4 KB). Value: Improves the performance of translation look-aside buffer lookup. The overhead for paging in and out is effectively eliminated providing a significant boost in performance.		Yes	If the VNF uses the DPDK, huge pages need to be configured for best performance.
Support for CMT and CAT	Ability to monitor how much LLC resources a VNF needs via CMT. Enables deterministic behavior of VMs and platform services such as virtual switch via CAT	CAT and CMT	Not yet enabled (on the roadmap)	Not yet enabled (on the roadmap)
Trusted Compute Pools	Supports security hardening by designating OpenStack compute pools to use hardware-based security features, such as Intel® TXT. Value: Provides the ability for cloud subscribers to request services run only on verified compute nodes.	Intel, TXT, trusted platform module (TPM)	Yes, If Intel TXT and TPM are enabled in BIOS	Open CIT client installed

Appendix A: Summary of EPA Features

EPA FEATURE	DESCRIPTION	RELATED PLATFORM CAPABILITIES	SUPPORTED BY NFVI	VNF / VNF MANAGER SUPPORT REQUIREMENT
OvS Firewall Driver	OS-agnostic firewall, fully integrated with OvS. Outperforms the stock iptables firewall used with a "vanilla" OvS.		Yes	N/A
Support for OVS-DPDK Controlled by OpenStack Networking	Allows hosts with OVS-DPDK to be controlled and configured by setting datapath type for bridges and creating vHost-user ports instead of tap or veth interfaces. Value: Accelerated performance and low-latency virtual switch.	DPDK	Yes	Yes
Support for OVS-DPDK controlled by OpenDaylight*	Allows OpenDaylight to manage OVS-DPDK and define the OpenFlow rules. Value: OpenDaylight controller can be used to control flows in the network.	DPDK	Yes	Yes
Telemetry Capture (via collectd)	Integration of OpenStack telemetry services with highly scalable monitoring system that collects statistics about the underlying hardware. Value: Enables more informed decisions for automatic provisioning of resources to be made.		Yes	N/A

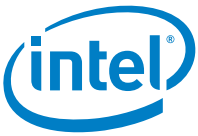
Appendix B: References

REFERENCE	SOURCE
Enhanced Platform Awareness for PCIe* Devices	https://wiki.openstack.org/wiki/Enhanced-platform-awareness-pcie
EPA training videos	https://builders.intel.com/university/networkbuilders/course/openstack-enhanced-platform-awareness-101 https://builders.intel.com/university/networkbuilders/course/openstack-enhanced-platform-awareness-102
Open Cloud Integrity Technology	https://01.org/opencit
OVS-DPDK	https://github.com/openstack/networking-ovs-dpdk/blob/master/doc/source/getstarted/devstack/ubuntu.rst https://01.org/openstack/blogs/stephenfin/2016/enabling-ovs-dpdk-openstack
Intel® Xeon® processor D-1500 Family	https://www.intel.com/content/www/us/en/embedded/products/broadwell-de/overview.html
Intel® Xeon® processor E5-2600 v3 product family	http://ark.intel.com/products/family/78583/Intel-Xeon-Processor-E5-v3-Family#@All
Intel® Xeon® processor E5-2600 v4 product family	http://ark.intel.com/products/family/91287/Intel-Xeon-Processor-E5-v4-Family#@All
SR-IOV Configuration Guide	http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf

Appendix C: Abbreviations

ABBREVIATION	DESCRIPTION
Intel® AES-NI	Intel® Advanced Encryption Standard New Instructions
API	Application Programming Interface
Intel® AVX	Intel® Advanced Vector Extensions
BIOS	Basic I/O System
CAT	Cache Allocation Technology
CLI	Command Line Interface
CMT	Cache Monitoring Technology
CLOS	Class of Service
CPU	Central Processing Unit
DPDK	Data Plane Development Kit
EPA	Enhanced Platform Awareness
ETSI	European Telecommunications Standards Institute
GRUB	Grand Unified Bootloader
Intel® HT Technology	Intel® Hyper-Threading Technology
I/O	Input/Output
IOMMU	Input-Output Memory Management Unit
LAN	Local Area Network
LLC	Last-Level Cache
MANO	Management and Orchestration
ML2	Modular Layer 2
NFV	Network Functions Virtualization
NFVI	NFV Infrastructure
NIC	Network Interface Card

ABBREVIATION	DESCRIPTION
NUMA	Non-Uniform Memory Access
Open CIT	Open Cloud Integrity Technology
OPNFV	Open Platform for NFV
OS	Operating System
OvS	Open vSwitch
OVSDB	Open vSwitch database
OVS-DPDK	DPDK-Accelerated Open vSwitch
PCI	Peripheral Component Interconnect
PCIe	PCI Express
pCPU	Physical CPU
PF	Physical Function
SHVS	Standard, High-Volume Servers
SIMD	Single Instruction, Multiple Data
SKU	Stock Keeping Unit
SR-IOV	Single Root I/O Virtualization
Intel® SSE4.2	Intel® Streaming SIMD Extensions 4.2
TPM	Trusted Platform Module
Intel® TXT	Intel® Trusted Execution Technology
vCPU	Virtual CPU
VF	Virtual Function
VIM	Virtualized Infrastructure Manager
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtualized Network Function
VxLAN	Virtual eXtensible LAN



 0817/MH/ICMCSW/PDF002 335372-001US