REFERENCE ARCHITECTURE

Intel Corporation

# Container Bare Metal Reference Architecture (BMRA) for 2nd Generation Intel® Xeon® Scalable Processor


## Authors

Louise Daly

Sreemanti Ghosh

Abdul Halim

Przemyslaw Lal

Gary Loughnane

David Lu

Dana Nehama


## 1 Introduction

The goal of this guide is to help developers adopt and use Enhanced Platform Awareness (EPA), advanced networking technologies, and device plugin features in a container bare metal reference architecture. This version of the paper has been updated to include coverage for the 2nd generation Intel® Xeon® Scalable processors (formerly codenamed Cascade Lake). The container Bare Metal Reference Architecture (BMRA) represents a baseline configuration of components that are combined to achieve optimal system performance for applications running in a container-based environment.

This document is part of the Network Transformation Experience Kit, which is available at: https://networkbuilders.intel.com/

Containers are one of the hottest technologies in cloud computing and fundamental to Cloud Native adoption. Moreso than the virtual machine, a container is lightweight, agile and portable. It can be quickly created, updated, and removed. Kubernetes* is the leading open source system for automating deployment, scaling, and management of containerized applications. To enhance Kubernetes for network functions virtualization (NFV) and networking usage, Intel and its partners are developing the following:

- Enhanced Platform Awareness suite of capabilities and methodologies that exposes Intel® Architecture platform features for increased and deterministic application and network performance.
- Networking features in Kubernetes including Multus, a plugin that provides multiple network interfaces within a Kubernetes container, SR-IOV CNI plugin, and Userspace CNI plugin to improve data throughput.
- Device plugin features in Kubernetes including GPU, FPGA, Intel® QuickAssist Technology, and SR-IOV device plugins to boost performance and platform efficiency.

# Table of Contents


**1 Introduction ... 1**
1.1 Technology overview ... 3
1.2 Terminology ... 3
1.3 Reference documents ... 4

**2 Physical and Software topology ... 5**
2.1 Physical topology ... 5
2.2 BMRA 2.0 Software Architecture ... 6

**3 Hardware BOM ... 6**

**4 Software BOM ... 7**
4.1 Platform BIOS settings ... 8

**5 System prerequisites ... 8**
5.1 Master and minion BIOS prerequisites ... 8
5.2 Master and minion nodes network interface requirements ... 9
5.3 Ansible Host, master, and minion software prerequisites ... 9

**6 Deploy Intel Bare Metal Reference Architecture using BMRA 2.0 Ansible Playbook ... 9**
6.1 Get Ansible Playbook and modify variables ... 9
6.2 Execute the Ansible playbook ... 12

**7 Post-deployment verification ... 13**
7.1 Check the Kubernetes cluster ... 13
7.2 Check Node Feature Discovery (NFD) ... 15
7.3 Check CPU Manager for Kubernetes ... 17
7.4 Intel Device Plugins for Kubernetes ... 18
7.4.1 SR-IOV network device plugin ... 18
7.4.2 Check Intel® QAT device plugin ... 19
7.5 Check Advanced Networking Features ... 19
7.5.1 Multus CNI plugin ... 19
7.5.2 SR-IOV CNI plugin ... 19
7.5.3 Userspace CNI plugin ... 19

**8 Use cases ... 20**
8.1 Use NFD to select the node to deploy the pod ... 20
8.2 Pod using SR-IOV ... 20
8.2.1 Pod using SR-IOV DPDK ... 20
8.2.2 Pod using SR-IOV CNI plugin ... 21
8.3 Pod using Userspace CNI plugin ... 23
8.3.1 Pod using Userspace CNI with OVS-DPDK ... 23
8.4 Pod using CPU Pinning and Isolation in Kubernetes ... 28

**9 Conclusion ... 28**


# Figures


Figure 1. Reference architecture topology ... 5
Figure 2. BMRA 2.0 software architecture ... 6


# Tables


Table 1. Terminology ... 3
Table 2. Reference documents ... 4
Table 3. Hardware BOM ... 6
Table 4. Software BOM ... 7
Table 5. Platform BIOS settings ... 8
Table 6. Group variables ... 10
Table 7. Host variables ... 12

## 1.1 Technology overview

Enhanced Platform Awareness (EPA) for Kubernetes represents a methodology and a related suite of changes across multiple layers of the orchestration stack targeting intelligent platform capability, configuration, and capacity data consumption. Specifically, EPA underpins the three-fold objective of discovery, scheduling, and isolation of server hardware capabilities. Intel and partners have worked together to progress this strategy further through the following technologies:

- Node Feature Discovery (NFD) enables generic hardware capability discovery in Kubernetes, including Intel® Xeon® processor-based hardware.
- CPU Manager for Kubernetes provides a mechanism for CPU core pinning and isolation of containerized workloads.
- Huge page support, added to Kubernetes v1.8, enables the discovery, scheduling and allocation of huge pages as a native first-class resource. This support addresses low latency and deterministic memory access requirements.
- SR-IOV provides I/O virtualization that makes a single PCIe device (typically a NIC) appear as many network devices in the Linux* kernel. In Kubernetes this results in network connections that can be separately managed and assigned to different pods.

One of the important parts in the Advanced Networking Features is Multus, which supports multiple network interfaces per pod to expand the networking capability of Kubernetes. Supporting multiple network interfaces is a key requirement for many virtual network functions (VNFs), as they require separation of control, management, and data planes. Multiple network interfaces are also used to support different protocols or software stacks and different tuning and configuration requirements.

Advanced Networking Features also introduced the SR-IOV CNI plugin and Userspace CNI plugin to enable high performance networking for container-based applications. The SR-IOV CNI plugin allows a Kubernetes pod to be attached directly to a SR-IOV virtual function (VF) using the standard SR-IOV VF driver in the container host's kernel. The Userspace CNI plugin was designed to implement userspace networking (as opposed to kernel space networking), like DPDK based applications. It can run with vSwitches such as OVS-DPDK or VPP and provides a high performance container networking solution through dataplane acceleration in NFV environments.

With the Kubernetes Device Plugin Framework, Intel provides several device plugins to free up CPU cycles and boost performance. It can deliver efficient acceleration of graphics, compute, data processing, security, and compression. These device plugins include:

- GPU device plugin: VNFs can take advantage of storing, streaming, and transcoding with the Intel GPU device plugin. Intel® Graphics Technology and Intel® Quick Sync Video Technology can accelerate graphics performance.
- FPGA device plugin: Scalable and programmable acceleration in a broad array of applications such as communications, data center, military, broadcast, automotive, and other end markets.
- Intel® QuickAssist Technology (Intel® QAT) device plugin: Directs crypto and data compression functionality to dedicated hardware, accelerating bulk crypto, public key encryption, and compression on Intel® architecture platforms.
- SR-IOV device plugin: supports DPDK VNFs that execute the VF driver and network protocol stack in userspace.

## 1.2 Terminology

**Table 1. Terminology**

| ABBREVIATION | DESCRIPTION |
|---|---|
| BIOS | Basic Input / Output System |
| BMRA | Bare Metal Reference Architecture |
| CNI | Container Networking Interface |
| DHCP | Dynamic Host Configuration Protocol |
| DPDK | Data Plane Development Kit |
| HA | High Availability |
| IA | Intel® Architecture |
| Intel® HT Technology | Intel® Hyper-Threading Technology |
| Intel® QAT | Intel® QuickAssist Technology |
| Intel® VT-d | Intel® Virtualization Technology (Intel® VT) for Directed I/O |
| Intel® VT-x | Intel® Virtualization Technology (Intel® VT) for IA-32, Intel® 64 and Intel® Architecture |
| K8s | Kubernetes* |
| NFD | Node Feature Discovery |
| NFV | Network Functions Virtualization |
| OS | Operating System |

| ABBREVIATION | DESCRIPTION |
|---|---|
| OVS DPDK | Open vSwitch with DPDK |
| RA | Reference Architecture |
| SA | Service Assurance |
| SDN | Software-Defined Networking |
| SHVS | Standard High-Volume Servers |
| SOCKS | Socket Secure |
| SR-IOV | Single Root Input/Output Virtualization |
| VLAN | Virtual LAN |
| VNF | Virtual Network Function |
| VPP | Vector Packet Processing |
| VXLAN | Virtual Extensible LAN |

## 1.3 Reference documents

**Table 2. Reference documents**

| REFERENCE | SOURCE |
|---|---|
| Advanced Networking Features in Kubernetes and Container Bare Metal Application Note | https://builders.intel.com/docs/networkbuilders/adv-network-features-in-kubernetes-app-note.pdf |
| Node Feature Discovery Application Note | https://builders.intel.com/docs/networkbuilders/node-feature-discovery-application-note.pdf |
| CPU Pinning and Isolation in Kubernetes Application Note | https://builders.intel.com/docs/networkbuilders/cpu-pin-and-isolation-in-kubernetes-app-note.pdf |
| Intel Device Plugins for Kubernetes Application Note | https://builders.intel.com/docs/networkbuilders/intel-device-plugins-for-kubernetes-appnote.pdf |

# 2 Physical and Software topology

## 2.1 Physical topology

Figure 1 shows the topology for the reference architecture described in this document.

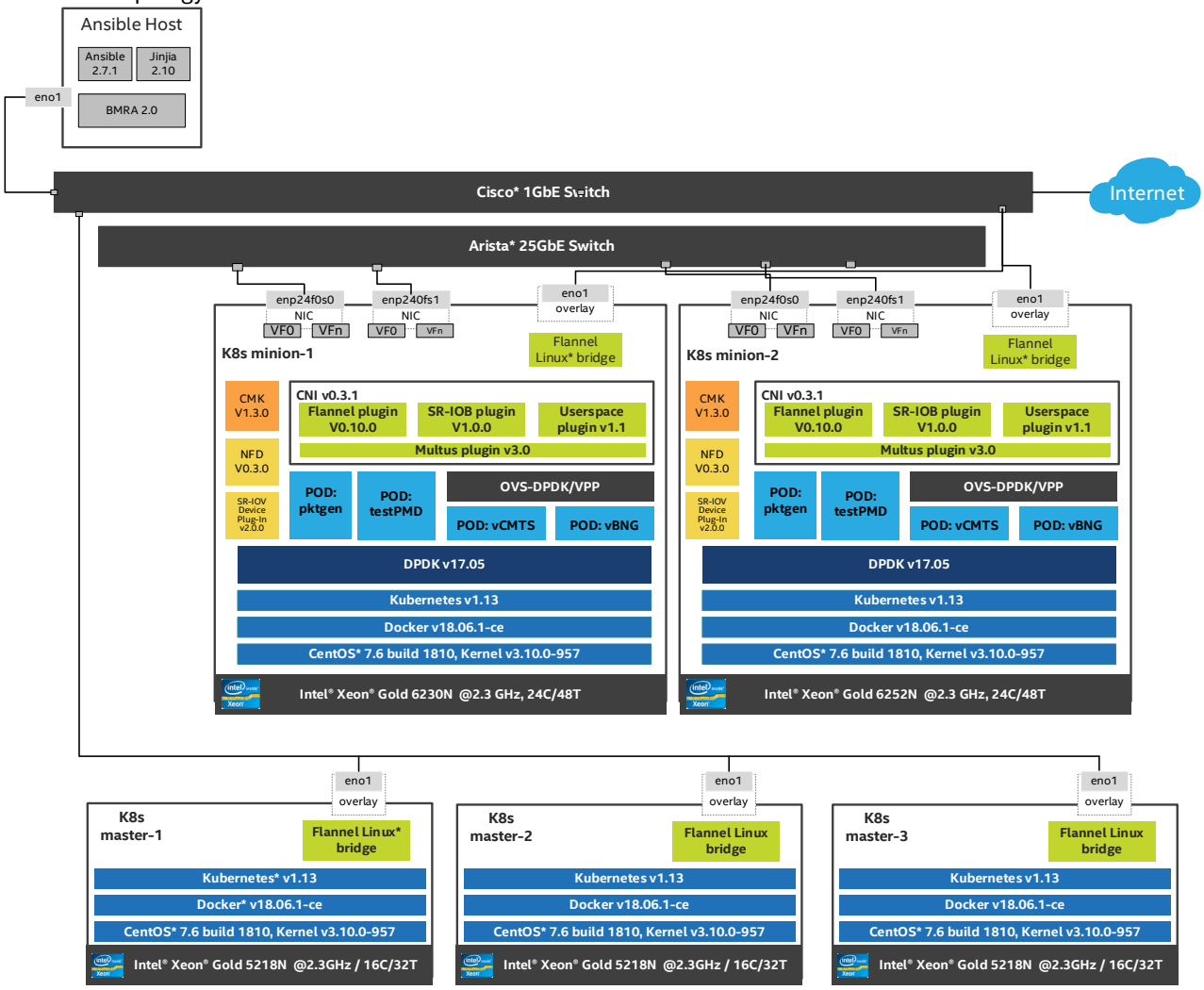



**Figure 1. Reference architecture topology**

## 2.2 BMRA 2.0 Software Architecture

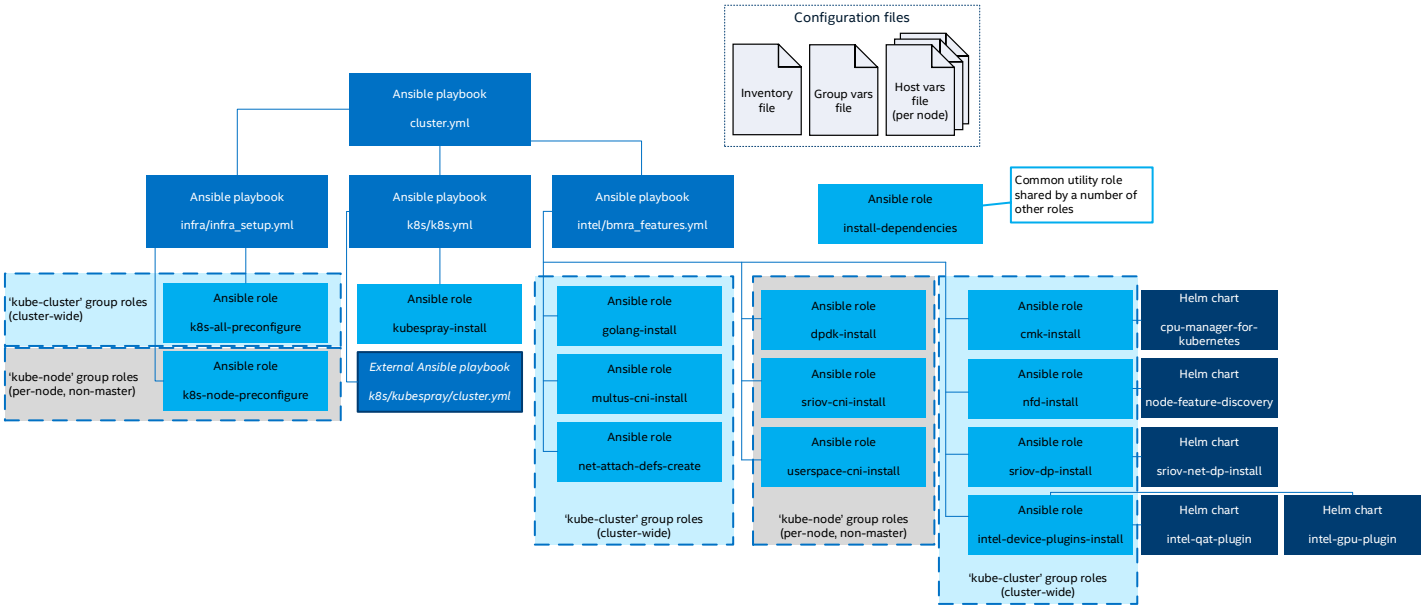



**Figure 2. BMRA 2.0 software architecture**

# 3 Hardware BOM

This section lists the hardware components and systems that were utilized in this reference architecture. 2nd Generation Intel® Xeon® Scalable processors feature a scalable, open architecture designed for the convergence of key workloads such as applications and services, control plane processing, high-performance packet processing, and signal processing.

**Table 3. Hardware BOM**

| ITEM | DESCRIPTION | NOTES |
|---|---|---|
| Platform | Intel® Xeon® Processor Scalable Family | Intel® Xeon® processor-based dual-processor server board<br>2 x 25 GbE integrated LAN ports |
| Processors | 6x Intel® Xeon® Gold 5218N Processor | 16 cores, 32 threads, 2.3 GHz, 105 W, 38.5 MB L3 total cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels |
| | 2x Intel® Xeon® Gold 6230N Processor | 20 cores, 40 threads, 2.0 GHz, 125 W, 27.5 MB L3 cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels |
| | 2x Intel® Xeon® Gold 6252N Processor | 24 cores, 48 threads, 2.3 GHz, 150 W, 22 MB L3 cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels |
| Memory | 192GB (12 x 16GB 2666MHz DDR RDIMM) or minimum all 6 memory channels populated (1 DPC) to achieve 384 GB | 192GB to 384GB |
| Networking | 2 x NICs - Required<br>Each NIC NUMA aligned | 2 x Dual Port 25GbE Intel® Ethernet Network Adapter XXV710 SFP28+ |
| | | 2 x Dual Port 10GbE Intel® Ethernet Converged Network Adapter X710 |
| | | 2 x Intel® Ethernet Server Adapter X520-DA2 SFP |
| Local Storage | 2 x >=480GB Intel® SSD SATA or Equivalent Boot Drive. This is for the primary Boot / OS storage. These drives can be sourced by the PCH. These drives should be capable of use in a RAID1 configuration. | 2 x Intel® NVMe P4510 Series 2.0TB each Drive recommended NUMA aligned - Required |
| Intel® QuickAssist Technology | Intel® C620 Series Chipset Integrated on baseboard<br>Intel® C627/C628 Chipset | Integrated w/NUMA connectivity to each CPU or minimum 16 Peripheral Component Interconnect express* (PCIe*) lane Connectivity to one CPU |

| ITEM | DESCRIPTION | NOTES |
|---|---|---|
| BIOS | Intel Corporation SE5C620.86<br>B.0D.01.0241<br>Release Date: 11/19/2018 | Intel® Hyper-Threading Technology (Intel® HT Technology) enabled<br>Intel® Virtualization Technology (Intel® VT-x) enabled<br>Intel® Virtualization Technology for Directed I/O (Intel® VT-d) enabled |
| Switches | Cisco* Catalyst 2960-XR<br>Arista* DCS-7280QR-C36-R | Cisco 1GbE Switch<br>Arista 25GbE Switch |

# 4 Software BOM

**Table 4. Software BOM**

| SOFTWARE FUNCTION | SOFTWARE COMPONENT | LOCATION |
|---|---|---|
| Host OS | CentOS* 7.6 build 1810<br>Kernel version: 3.10.0-957.1.3.el7.x86_64 | https://www.centos.org/ |
| Ansible* | Ansible v2.7.1 | https://www.ansible.com/ |
| BMRA 2.0 Ansible Playbook | Master Playbook v1.0 | https://github.com/intel/container-experience-kits |
| Python* | Python 2.7 | https://www.python.org/ |
| Kubespray* | Kubespray: v2.8.0-31-g3c44ffc | https://github.com/kubernetes-sigs/kubespray |
| Docker* | Docker* 18.06.1-ce, build e68fc7a | https://www.docker.com/ |
| Container orchestration engine | Kubernetes* v1.13.0 | https://github.com/kubernetes/kubernetes |
| CPU Manager for Kubernetes* | CPU Manager for Kubernetes* v1.3.0 | https://github.com/intel/CPU-Manager-for-Kubernetes |
| Node Feature Discovery | NFD v0.3.0 | https://github.com/kubernetes-sigs/node-feature-discovery |
| Data Plane Development Kit | DPDK 17.05.0 | http://dpdk.org/git/dpdk |
| Open vSwitch with DPDK | OVS-DPDK 2.11.90 | http://docs.openvswitch.org/en/latest/intro/install/dpdk/ |
| Vector Packet Processing | VPP 19.01 | https://docs.fd.io/vpp/19.01/index.html |
| Multus CNI | Multus CNI v4.0 | https://github.com/intel/multus-cni |
| SR-IOV CNI | SR-IOV CNI v1.0 | https://github.com/intel/SR-IOV-network-device-plugin |
| Userspace CNI | Userspace CNI v1.0 | https://github.com/intel/userspace-cni-network-plugin |
| Intel Ethernet Drivers | | https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1<br>https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1<br>https://sourceforge.net/projects/e1000/files/i40e%20stable/2.0.30<br>https://sourceforge.net/projects/e1000/files/i40evf%20stable/2.0.30 |

## 4.1 Platform BIOS settings

**Table 5. Platform BIOS settings**

| MENU (ADVANCED) | PATH TO BIOS SETTING | BIOS SETTING | SETTINGS FOR DETERMINISTIC PERFORMANCE | SETTINGS FOR MAX PERFORMANCE WITH TURBO MODE ENABLED | REQUIRED OR RECOMMENDED |
|---|---|---|---|---|---|
| Power Configuration | CPU P State Control | EIST PSD Function | HW_ALL | SW_ALL | *Recommended* |
| | | Boot Performance Mode | Max. Performance | Max. Performance | *Required* |
| | | Energy Efficient Turbo | Disable | Disable | *Recommended* |
| | | Turbo Mode | Disable | Enable | *Recommended* |
| | | Intel® SpeedStep® (Pstates) Technology | Disable | Enable | *Recommended* |
| | Hardware PM State Control | Hardware P-States | Disable | Disable | *Recommended* |
| | CPU C State Control | Autonomous Core C-State | Disable | Enable | *Recommended* |
| | | CPU C6 Report | Disable | Disable | *Recommended* |
| | | Enhanced Halt State (C1E) | Disable | Enable | *Recommended* |
| | Energy Perf Bias | Power Performance Tuning | BIOS Controls EPB | BIOS Controls EPB | *Recommended* |
| | | ENERGY_PERF_BIAS_CFG Mode | Perf | Perf | *Recommended* |
| | Package C State Control | Package C State | C0/C1 State | C6 | *Recommended* |
| Intel® Ultra Path Interconnect (Intel® UPI) Configuration | Intel® UPI General Configuration | LINK L0P ENABLE | Disable | Disable | *Recommended* |
| | | LINK L1 ENABLE | Disable | Disable | *Recommended* |
| | | SNC | Disable | Disable | *Recommended* |
| Memory Configuration | | Enforce POR | Disable | Disable | *Recommended* |
| | | IMC Interleaving | 2-Way Interleave | 2-Way Interleave | *Recommended* |
| | | Volatile Memory Mode | 2 LM mode | 2 LM mode | *Required* |
| | | Force 1-Ch Way in FM | Disabled | Disabled | *Required* |
| Platform Configuration | Miscellaneous Configuration | Serial Debug Message Level | Minimum | Minimum | *Recommended* |
| | PCI Express* Configuration | PCIe* ASPM Support | Per Port | Per Port | *Recommended* |
| Uncore | Uncore Frequency Scaling | | Disable | Disable | *Required* |

***Note:*** To gather performance data required for conformance, use either column with deterministic performance or turbo mode enabled in this table. Some solutions may not provide the BIOS options that are documented in this table. For Intel® Select Solution, the BIOS should be set to the “Max Performance” profile with Virtualization.

# 5 System prerequisites

This section describes the minimal system prerequisites needed for the Ansible Host and master/minion nodes.

## 5.1 Master and minion BIOS prerequisites

Enter the BIOS menu and update the configuration as follows:

| PACKAGE | ANSIBLE HOST | MASTER/MINION |
|---|---|---|
| Intel® VT-x | Enabled | Enabled |
| Intel® HT Technology | Enabled | Enabled |
| Intel® VT-d (SR-IOV) | Enabled | Enabled |

## 5.2 Master and minion nodes network interface requirements

The following list provides a brief description of different networks and network interfaces used in the lab setup.

- Internet network
  - Ansible Host accessible
  - Capable of downloading packages from the internet
  - Can be configured for Dynamic Host Configuration Protocol (DHCP) or with static IP address
- Management network and flannel pod network interface
  - Kubernetes master and minion nodes inter-node communications
  - Flannel pod network will run over this network
  - Configured to use a private static address
- Tenant data network(s)
  - Dedicated networks for traffic
  - SR-IOV enabled
  - VF can be DPDK bound in pod

## 5.3 Ansible Host, master, and minion software prerequisites

Enter the following commands in Ansible Host:

```
# sudo yum install epel-release
# sudo yum install ansible
# easy_install pip
# pip2 install jinja2 –upgrade
# sudo yum install python36 –y
```

Enable passwordless login between all nodes in the cluster.

Step 1: Create Authentication SSH-Kegen Keys on Ansible Host:

```
# ssh-keygen
```

Step 2: Upload Generated Public Keys to all the nodes from Ansible Host:

```
# ssh-copy-id root@node-ip-address
```

# 6 Deploy Intel Bare Metal Reference Architecture using BMRA 2.0 Ansible Playbook

## 6.1 Get Ansible Playbook and modify variables

1. Get Ansible playbook.

```
# git clone https://github.com/intel/container-experience-kits.git
# cd container-experience-kits
```

2. Copy example inventory file to the playbook home location.

```
# cp examples/inventory.ini .
```

3. Edit the inventory.ini to reflect the requirement. Here is the sample file.

```
[all]
node1    ansible_host=10.250.250.161 ip=10.250.250.161
node2    ansible_host=10.250.250.162 ip=10.250.250.162
node3    ansible_host=10.250.250.163 ip=10.250.250.163
node4    ansible_host=10.250.250.166 ip=10.250.250.166
node5    ansible_host=10.250.250.167 ip=10.250.250.167

[kube-master]
node1
node2
node3

[etcd]
node1
node2
node3

[kube-node]
node5

[k8s-cluster:children]
kube-master
kube-node

[calico-rr]
```

4. Copy group_vars and host_vars directories to the playbook home location.

```
# cp -r examples/group_vars examples/host_vars .
```

5. Update group_vars to match the desired configuration, use Table 6 as a reference.

```
# vim group_vars/all.yml

---
## BMRA master playbook variables ##

# Node Feature Discovery
nfd_enabled: true
nfd_build_image_locally: true
nfd_namespace: kube-system
nfd_sleep_interval: 30s

# Intel CPU Manager for Kubernetes
cmk_enabled: true
cmk_namespace: kube-system
cmk_use_all_hosts: false # 'true' will deploy CMK on the master nodes too
#cmk_hosts_list: node1,node2 # allows to control where CMK nodes will run, leave this option
commented out to deploy on all K8s nodes
cmk_shared_num_cores: 2 # number of CPU cores to be assigned to the "shared" pool on each of
the nodes
cmk_exclusive_num_cores: 2 # number of CPU cores to be assigned to the "exclusive" pool on each
of the nodes
#cmk_shared_mode: packed # choose between: packed, spread, default: packed
#cmk_exclusive_mode: packed # choose between: packed, spread, default: packed

# Intel SRIOV Network Device Plugin
sriov_net_dp_enabled: true
sriov_net_dp_namespace: kube-system
# whether to build and store image locally or use one from public external registry
sriov_net_dp_build_image_locally: true

# Intel Device Plugins for Kubernetes
qat_dp_enabled: true
qat_dp_namespace: kube-system
gpu_dp_enabled: true
gpu_dp_namespace: kube-system

# Forces installation of the Multus CNI from the official Github repo on top of the Kubespray
built-in one
force_external_multus_installation: true

## Proxy configuration ##
proxy_env:
  http_proxy: ""
  https_proxy: ""
  no_proxy:
"localhost,127.0.0.1,.intel.com,10.254.0.1,10.250.250.161,10.250.250.162,10.250.250.163,10.250.
250.166,10.250.250.167"

## Kubespray variables ##

# default network plugins and kube-proxy configuration
kube_network_plugin_multus: true
multus_version: v3.2
```

***Note:*** Please pay special attention to the 'http_proxy', 'https_proxy' and 'no_proxy' variables if operated from behind proxy. Add all cluster node IP addresses to 'no_proxy' variable.

**Table 6. Group variables**

| VARIABLE | CHOICES/DEFAULTS | DESCRIPTION |
|---|---|---|
| nfd_enabled | Boolean, default "true" | Install NFD on the target cluster |
| nfd_build_image_locally | Boolean, default "true" | Build and host NFD image in the cluster local Docker registry, set to false to use NFD image from quay.io |
| nfd_namespace | String, default "kube-system" | Kubernetes namespace to be used for NFD deployment |
| cmk_enabled | Boolean, default "true" | Install CMK on the target cluster |
| cmk_namespace | String, default "kube-system" | Kubernetes namespace to be used for CMK deployment |

| VARIABLE | CHOICES/DEFAULTS | DESCRIPTION |
|---|---|---|
| cmk_use_all_hosts | Boolean, default “true” | Use --all-hosts argument for CMK cluster-init, effectively depoying CMK on all Kubernetes nodes, if set to “false”, cmk_hosts_list must be set |
| cmk_hosts_list | Comma separated list of node names | List of node names to deploy CMK on, ignored if cmk_use_all_hosts set to true |
| cmk_shared_num_cores | Integer, default 2 | Number of CPU cores to be added to the shared pool |
| cmk_exclusive_num_cores | Integer, default: 2 | Number of CPU cores to be added to the exclusive pool |
| cmk_shared_mode | “packed” or “shared”, default: “packed” | Allocation mode for shared pool (how cores are allocated across NUMA nodes) |
| cmk_exclusive_mode | “packed” or “shared”, default: “packed” | Allocation mode for exclusive pool (how cores are allocated across NUMA nodes) |
| sriov_net_dp_enabled | Boolean, default “true” | Install SRIOV Network Device Plugin on the target cluster |
| sriov_net_dp_namespace | String, default “kube-system” | Kubernetes namespace to be used for SRIOV Network Device Plugin deployment |
| sriov_net_dp_build_image_locally | Boolean, default “true” | Build and host SRIOV Network Device Plugin image in the cluster local Docker registry, set to false to use SRIOV Network Device Plugin image from Docker Hub |
| qat_dp_enabled | Boolean, default “true” | Install Intel QAT Device Plugin on the target cluster |
| qat_dp_namespace | String, default “kube-system” | Kubernetes namespace to be used for Intel QAT Device Plugin deployment |
| gpu_dp_enabled | Boolean, default “true” | Install Intel GPU Device Plugin on the target cluster |
| gpu_dp_namespace | String, default “kube-system” | Kubernetes namespace to be used for Intel GPU Device Plugin deployment |
| example_net_attach_defs:<br>sriov_net_dp<br>userspace_vpp<br>userspace_ovs_dpdk | Boolean for values, default: “true” | Create example net-attach-def objects to be used with the Userspace CNI Plugin and SRIOV Network Device Plugin |
| proxy_env<br>http_proxy<br>https_proxy<br>no_proxy | Dictionary, values for keys - <address>:<port> for http_proxy and https_proxy, list of addresses and CIDRs for no_proxy | Proxy configuration, comment out if you’re not behind proxy, add all K8s nodes IP addresses to the “no_proxy” key |
| kube_network_plugin_multus | Boolean, default “true” | Kubespray variable, don’t change |
| multus_version | Git tag, default “v3.2” | Kubespray variable, change if you specifically need to use different Multus version |
| kube_network_plugin | String - plugin name, default “flannel” | Kubespray variable, don’t change |
| kube_pods_subnet | CIDR string, default “10.244.0.0/16” | Kubespray variable |
| kube_proxy_mode | String, default “iptables” | Kubespray variable |
| helm_enabled | Boolean, default“true” | Kubespray variable |
| registry_enabled | Boolean, default“true” | Kubespray variable |
| registry_storage_class | String, default“null” | Kubespray variable |
| registry_local_address | <address>:<port>, e.g. "localhost:5000" | Kubespray variable |

6. Update files in the host_vars directory to match the desired configuration. Use Table 7 as a reference.

***Note:*** The host_vars folder should contain individual .yml files for all the nodes defined in the inventory file. In the examples we have defined 2 nodes, thus host_vars should contain the respective .yml files for the nodes. For e.g. node1.yml and node2.yml.

***Note:*** The Isolate CPUs from kernel scheduler variable ‘isolcpus’ should be assigned a number which represents the number of CPUs of the cluster node that we want to allocate.

**Table 7. Host variables**

| VARIABLE | CHOICES/DEFAULTS | DESCRIPTION |
|---|---|---|
| sriov_enabled | Boolean | Enables SRIOV VFs for interfaces specified in "sriov_nics" array |
| sriov_nics | Array of strings, e.g. ["eth1", "eth2"] | Network interfaces names of the SRIOV PFs |
| sriov_numvfs | Integer | Number of VFs to be created per PF |
| sriov_cni_enabled | Boolean | Whether to install SRIOV CNI plugin on target node |
| sriov_net_dp_autogenerate | Boolean | If "true", config with a single resource pool and all SRIOV NICs from "sriov_nics" will be created, if "false" configuration from "sriov_net_dp_config" will be used |
| sriov_net_dp_device_type | String, "netdevice", "vfio" or "uio" | SRIOV device driver type |
| sriov_net_dp_sriov_mode | Boolean. default "false" | Whether the NICs support SR-IOV, "false" for VM deployments, "true" for bare metal installation |
| sriov_net_dp_config:<br>- resource_name<br>root_devices<br>device_type<br>sriov_mode | Array of dicts, please see description | Configure as described here:<br>https://github.com/intel/sriov-network-device-plugin#config-parameters<br>only if "sriov_net_dp_autogenerate" set to "false" |
| userspace_cni_enabled | Boolean, default "true" | Enables installation of Userspace CNI plugin on target machine |
| vpp_enabled | Boolean, default "true" | Enables installation of virtual userspace network switch, selecting one is recommended |
| ovs_dpdk_enabled | Boolean, default "true" | |
| force_nic_drivers_update | Boolean, default "true" | Update i40e and i40evf drivers on the target machine |
| hugepages_enabled | Boolean, default "true" | Enable hugepages support |
| default_hugepage_size | "2M" or "1G" | Default hugepage size |
| hugepages_1G | Integer | Sets how many hugepages of each size should be created |
| hugepages_2M | | |
| isolcpus_enabled | Boolean, default "true" | Isolates CPUs from Linux scheduler |
| isolcpus | Comma separated list of integers, maybe mixed with ranges,<br>e.g. "1,2,3-6,8-15,24" | List of CPUs to be isolated |

***Note:*** Not all variables are defined in the group_vars and host_vars directories - more fine-grained control is possible and can be achieved by overriding vars defined on the role level. Inspect files located in "roles/*/vars" and "roles/*/defaults" for more advanced configuration options, but remember - some of these variables are not exposed for a reason!

7. Update and initialize git submodule.

```
# git submodule update --init
```

This git repository has nested submodules to support kubespray installation. The submodule update command will recurse into registered submodules.

## 6.2 Execute the Ansible playbook

The Ansible playbook runs three sub playbooks. The sub playbooks are:

- Infra: This playbook modifies kernel boot parameters in the cluster. It sets up boot parameters as follows:
  - Uses isolcpus boot parameter to ensure exclusive cores in CPU Manager for Kubernetes are not affected by other system tasks.
  - Adds the isolcpus in grub file.
  - I/O Memory Management Unit (IOMMU) support is not enabled by default in the CentOS* 7.0 distribution, however, IOMMU support is required for a VF to function properly when assigned to a virtual environment, such as a VM or a container.
  - Enables IOMMU support for Linux kernels.
  - Specifies 1G hugepage size explicitly. (For hugepage sizes other than 2M, the size must be specified explicitly. Optionally, it can be set as the default hugepage size for the system)
  - SR-IOV virtual functions are created by writing an appropriate value to the sriov_numvfs parameter Appropriate SR-IOV PCI address must be used to fit the environment.
  - Sets the firewall rules on all minion nodes:

- K8s: Kubespray is a project under the Kubernetes community umbrella that helps deploy production-ready Kubernetes clusters easily. For details, refer to: https://kubespray.io. This playbook deploys a high availability (HA) cluster using Kubespray.
- Intel: This playbook deploys Enhanced Platform Awareness, advanced networking technologies and device plugin features as follows. These plugins are deployed using helm chart.
  - Deploys Node Feature Discovery which is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform that can, in turn, be used to facilitate intelligent scheduling of a workload.
  - Deploys CPU Manager for Kubernetes which performs a variety of operations to enable core pinning and isolation on a container or a thread level
  - Deploys SR-IOV network device plugin which discovers and exposes SR-IOV network resources as consumable extended resources in Kubernetes
  - Deploys Multus CNI plugin which is specifically designed to provide support for multiple networking interfaces in a Kubernetes environment.
  - Deploys SR-IOV CNI plugin which allow a Kubernetes pod to be attached directly to an SR-IOV virtual function (VF) using the standard SR-IOV VF driver in the container host's kernel.
  - Deploys Userspace CNI plugin which is a Container Network Interface plugin designed to implement user space networking such as DPDK based applications.
  - Deploys Quick Assist Device plugin in which a workload can alleviate pipeline bottlenecks and improve performance by offloading cryptographic operations of user traffic to a Quick Assist device.

Deploy Intel Bare Metal Reference Architecture with the following commands:

```
# ansible-playbook -i inventory.ini playbooks/cluster.yml
```

# 7 Post-deployment verification

This section shows how to verify all the components deployed by BMRA 2.0 scripts.

## 7.1 Check the Kubernetes cluster

1. Check the post-deployment node status of master & minion:

```
# kubectl get nodes -o wide
NAME    STATUS   ROLES    AGE     VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION               CONTAINER-RUNTIME
node1   Ready    master   4d12h   v1.13.5   10.250.250.161   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
node2   Ready    master   4d12h   v1.13.5   10.250.250.162   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
node3   Ready    master   4d12h   v1.13.5   10.250.250.163   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
node4   Ready    node     4d12h   v1.13.5   10.250.250.166   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
node5   Ready    node     4d12h   v1.13.5   10.250.250.167   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
node6   Ready    node     4d12h   v1.13.5   10.250.250.168   <none>        CentOS Linux 7
(Core)   3.10.0-957.12.2.el7.x86_64   docker://18.6.2
```

2. Check pod status of master & minion. All Pod states should be in Running or Completed status.

```
# kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY   STATUS
RESTARTS   AGE
kube-system   cmk-cmk-cluster-init-pod                      0/1     Completed          0
4d12h
kube-system   cmk-init-install-discover-pod-node4           0/2     Completed          0
4d12h
kube-system   cmk-init-install-discover-pod-node5           0/2     Completed          0
4d12h
kube-system   cmk-init-install-discover-pod-node6           0/2     Completed          0
4d12h
kube-system   cmk-reconcile-nodereport-ds-node4-s7dzx       2/2     Running            2
4d12h
kube-system   cmk-reconcile-nodereport-ds-node5-6d926       2/2     Running            2
4d12h
kube-system   cmk-reconcile-nodereport-ds-node6-44sxk       2/2     Running            3
4d12h
kube-system   cmk-webhook-deployment-8456cc6f5-kjjfd        1/1     Running            1
4d12h
kube-system   coredns-644c686c9-b6b8s                       1/1     Running            1
4d10h
kube-system   coredns-644c686c9-nf5c4                       1/1     Running            1
4d10h
```

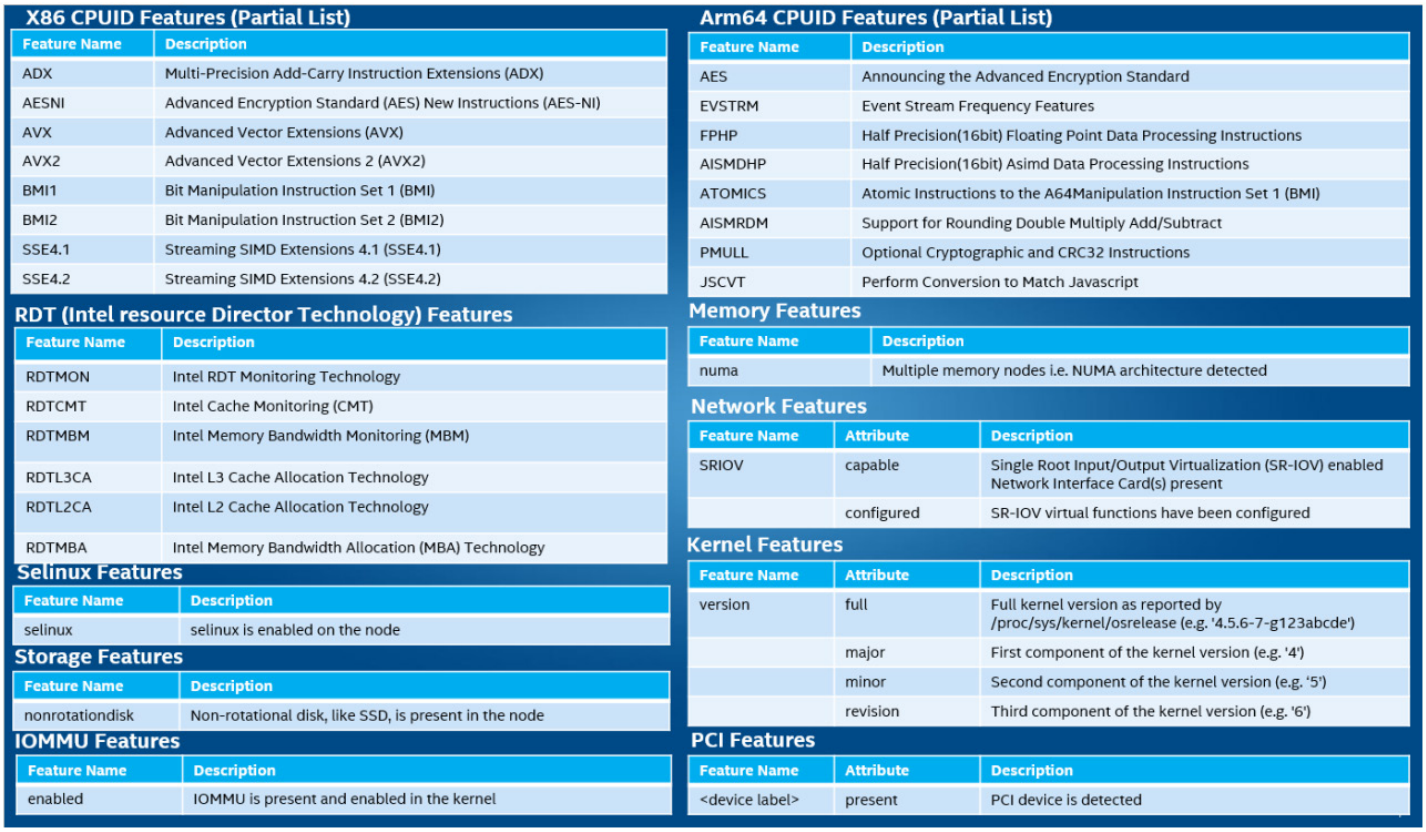
```
kube-system   dns-autoscaler-586f58b8bf-djf4q                  1/1     Running             2
4d12h
kube-system   intel-gpu-plugin-intel-gpu-plugin-5btnx          1/1     Running             1
4d12h
kube-system   intel-gpu-plugin-intel-gpu-plugin-dbh95          1/1     Running             1
4d12h
kube-system   intel-gpu-plugin-intel-gpu-plugin-psr9m          1/1     Running             1
4d12h
kube-system   intel-qat-plugin-intel-qat-plugin-6dgn8          1/1     Running
31          4d12h
kube-system   intel-qat-plugin-intel-qat-plugin-cmvxg          1/1     Running
30          4d12h
kube-system   intel-qat-plugin-intel-qat-plugin-qlgx9          1/1     Running
31          4d12h
kube-system   kube-apiserver-node1                            1/1     Running             3
4d12h
kube-system   kube-apiserver-node2                            1/1     Running             3
4d12h
kube-system   kube-apiserver-node3                            1/1     Running             4
4d12h
kube-system   kube-controller-manager-node1                   1/1     Running             3
4d12h
kube-system   kube-controller-manager-node2                   1/1     Running             2
4d12h
kube-system   kube-controller-manager-node3                   1/1     Running             4
4d12h
kube-system   kube-flannel-ds-amd64-55bbn                     1/1     Running             0
4d10h
kube-system   kube-flannel-ds-amd64-5bjrs                     1/1     Running
0          4d10h
kube-system   kube-flannel-ds-amd64-cwrdh                     1/1     Running             0
4d10h
kube-system   kube-flannel-ds-amd64-kcxfz                     1/1     Running             0
4d10h
kube-system   kube-flannel-ds-amd64-mfzwf                     1/1     Running
0          4d10h
kube-system   kube-flannel-ds-amd64-nkhxw                     1/1     Running
14          4d10h
kube-system   kube-multus-ds-amd64-68zhz                      1/1     Running             0
4d10h
kube-system   kube-multus-ds-amd64-9c989                      1/1     Running             0
4d10h
kube-system   kube-multus-ds-amd64-bwj7v                      1/1     Running             0
4d10h
kube-system   kube-multus-ds-amd64-pfsgz                      1/1     Running             0
4d10h
kube-system   kube-multus-ds-amd64-q59d2                      1/1     Running             0
4d10h
kube-system   kube-multus-ds-amd64-sg24p                      1/1     Running             0
4d10h
kube-system   kube-proxy-98jm8                                1/1     Running             1
4d10h
kube-system   kube-proxy-9n5mk                                1/1     Running             1
4d10h
kube-system   kube-proxy-gn94s                                1/1     Running             2
4d10h
kube-system   kube-proxy-nxrvs                                1/1     Running             2
4d10h
kube-system   kube-proxy-pzw9v                                1/1     Running             2
4d10h
kube-system   kube-proxy-qc22g                                1/1     Running             1
4d10h
kube-system   kube-scheduler-node1                            1/1     Running             2
4d12h
kube-system   kube-scheduler-node2                            1/1     Running             2
4d12h
kube-system   kube-scheduler-node3                            1/1     Running             4
4d12h
```

```
kube-system   kubernetes-dashboard-8457c55f89-kc56n              1/1     Running              3
4d12h
kube-system   metrics-server-cdd46d856-9lt4c                     2/2     Running              5
4d12h
kube-system   nfd-node-feature-discovery-5f8dg                   1/1     Running
12         4d12h
kube-system   nfd-node-feature-discovery-5m5km                   1/1     Running              9
4d12h
kube-system   nfd-node-feature-discovery-xjj5b                   1/1     Running
11         4d12h
kube-system   nginx-proxy-node4                                  1/1     Running              3
4d12h
kube-system   nginx-proxy-node5                                  1/1     Running              3
4d12h
kube-system   nginx-proxy-node6                                  1/1     Running              3
4d12h
kube-system   registry-kfchp                                     1/1     Running              2
4d12h
kube-system   registry-proxy-bptpx                               1/1     Running              2
4d12h
kube-system   registry-proxy-f6qlk                               1/1     Running              3
4d12h
kube-system   registry-proxy-lvt6z                               1/1     Running              2
4d12h
kube-system   sriov-net-dp-kube-sriov-device-plugin-amd64-9n422  1/1     Running              1
4d12h
kube-system   sriov-net-dp-kube-sriov-device-plugin-amd64-ngsps  1/1     Running              1
4d12h
kube-system   sriov-net-dp-kube-sriov-device-plugin-amd64-x4zpk  1/1     Running              1
4d12h
kube-system   tiller-deploy-dc85f7fbd-7fn6r                      1/1     Running              3
4d12h
```

## 7.2 Check Node Feature Discovery (NFD)

Node Feature Discovery (NFD) is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform that can, in turn, be used to facilitate intelligent scheduling of a workload. Node Feature Discovery is part of the Enhanced Platform Awareness (EPA) suite which represents a methodology and a set of changes in Kubernetes targeting intelligent configuration and capacity consumption of platform capabilities. NFD runs as a separate container on each individual node of the cluster, discovers capabilities of the node, and finally, publishes these as node labels using the Kubernetes API. NFD only handles non-allocatable features.

NFD currently detects the features shown in the following diagram.



**X86 CPUID Features (Partial List)**

| Feature Name | Description |
|---|---|
| ADX | Multi-Precision Add-Carry Instruction Extensions (ADX) |
| AESNI | Advanced Encryption Standard (AES) New Instructions (AES-NI) |
| AVX | Advanced Vector Extensions (AVX) |
| AVX2 | Advanced Vector Extensions 2 (AVX2) |
| BMI1 | Bit Manipulation Instruction Set 1 (BMI) |
| BMI2 | Bit Manipulation Instruction Set 2 (BMI2) |
| SSE4.1 | Streaming SIMD Extensions 4.1 (SSE4.1) |
| SSE4.2 | Streaming SIMD Extensions 4.2 (SSE4.2) |

**RDT (Intel resource Director Technology) Features**

| Feature Name | Description |
|---|---|
| RDTMON | Intel RDT Monitoring Technology |
| RDTCMT | Intel Cache Monitoring (CMT) |
| RDTMBM | Intel Memory Bandwidth Monitoring (MBM) |
| RDTL3CA | Intel L3 Cache Allocation Technology |
| RDTL2CA | Intel L2 Cache Allocation Technology |
| RDTMBA | Intel Memory Bandwidth Allocation (MBA) Technology |

**Selinux Features**

| Feature Name | Description |
|---|---|
| selinux | selinux is enabled on the node |

**Storage Features**

| Feature Name | Description |
|---|---|
| nonrotationdisk | Non-rotational disk, like SSD, is present in the node |

**IOMMU Features**

| Feature Name | Description |
|---|---|
| enabled | IOMMU is present and enabled in the kernel |

**Arm64 CPUID Features (Partial List)**

| Feature Name | Description |
|---|---|
| AES | Announcing the Advanced Encryption Standard |
| EVSTRM | Event Stream Frequency Features |
| FPHP | Half Precision(16bit) Floating Point Data Processing Instructions |
| AISMDHP | Half Precision(16bit) Asimd Data Processing Instructions |
| ATOMICS | Atomic Instructions to the A64Manipulation Instruction Set 1 (BMI) |
| AISMRDM | Support for Rounding Double Multiply Add/Subtract |
| PMULL | Optional Cryptographic and CRC32 Instructions |
| JSCVT | Perform Conversion to Match Javascript |

**Memory Features**

| Feature Name | Description |
|---|---|
| numa | Multiple memory nodes i.e. NUMA architecture detected |

**Network Features**

| Feature Name | Attribute | Description |
|---|---|---|
| SRIOV | capable | Single Root Input/Output Virtualization (SR-IOV) enabled Network Interface Card(s) present |
| | configured | SR-IOV virtual functions have been configured |

**Kernel Features**

| Feature Name | Attribute | Description |
|---|---|---|
| version | full | Full kernel version as reported by /proc/sys/kernel/osrelease (e.g. '4.5.6-7-g123abcde') |
| | major | First component of the kernel version (e.g. '4') |
| | minor | Second component of the kernel version (e.g. '5') |
| | revision | Third component of the kernel version (e.g. '6') |

**PCI Features**

| Feature Name | Attribute | Description |
|---|---|---|
| <device label> | present | PCI device is detected |

**Figure 3. Features detected by NFD**

To verify that NFD is running as expected, use the following command:

```
# kubectl get ds --all-namespaces | grep nfd-node-feature-discovery
kube-system   nfd-node-feature-discovery                    3         3         3       3
3           <none>                          4d12h
```

To check the labels created by NFD:

```
# kubectl label node --list --all
Listing labels for Node./node1:
 kubernetes.io/hostname=node1
 node-role.kubernetes.io/master=
 beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
Listing labels for Node./node2:
 kubernetes.io/hostname=node2
 node-role.kubernetes.io/master=
 beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
Listing labels for Node./node3:
 node-role.kubernetes.io/master=
 beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
 kubernetes.io/hostname=node3
Listing labels for Node./node4:
 node.alpha.kubernetes-incubator.io/nfd-network-sriov-configured=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-HTT=true
 beta.kubernetes.io/os=linux
 node.alpha.kubernetes-incubator.io/nfd-cpuid-SSE4.2=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-MMXEXT=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-FMA3=true
 node.alpha.kubernetes-incubator.io/nfd-network-sriov=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-SSE3=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-SSE4.1=true
 node.alpha.kubernetes-incubator.io/nfd-iommu-enabled=true
 kubernetes.io/hostname=node4
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBM=true
```

```
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBA=true
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTCMT=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-ERMS=true
.....
node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512DQ=true
 node.alpha.kubernetes-incubator.io/nfd-pstate-turbo=true
 node.alpha.kubernetes-incubator.io/nfd-network-sriov=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-MMX=true
 cmk.intel.com/cmk-node=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-MPX=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-SSE=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-RDSEED=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-HLE=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-CMOV=true
 beta.kubernetes.io/os=linux
 node.alpha.kubernetes-incubator.io/nfd-cpuid-RTM=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-SSE3=true
 beta.kubernetes.io/arch=amd64
 node.alpha.kubernetes-incubator.io/nfd-cpuid-HTT=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-RDRAND=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-ADX=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-BMI1=true
 node.alpha.kubernetes-incubator.io/nfd-network-sriov-configured=true
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBA=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-FMA3=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX2=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-BMI2=true
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTCMT=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512BW=true
 node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBM=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-RDTSCP=true
```

## 7.3 Check CPU Manager for Kubernetes

Kubernetes supports CPU and memory first class resources, while also providing basic support for CPU Pinning and Isolation through the native CPU Manager. To aid commercial adoption, Intel has created CPU Manager for Kubernetes, an open source project that introduces additional CPU optimization capabilities. Without CPU Manager for Kubernetes, the kernel task scheduler will treat all CPUs as available for scheduling process threads and regularly preempts executing process threads to give CPU time to other threads. This non-deterministic behavior makes it unsuitable for latency sensitive workloads.

Using the preconfigured isolcpus boot parameter, CPU Manager for Kubernetes can ensure that a CPU (or set of CPUs) is isolated from the kernel scheduler. Then the latency sensitive workload process thread(s) can be pinned to execute on that isolated CPU set only, providing them exclusive access to that CPU set. While beginning to guarantee the deterministic behavior of priority workloads, isolating CPUs also addresses the need to manage resources, which allows multiple VNFs to coexist on the same physical server. The exclusive pool within CPU Manager for Kubernetes assigns entire physical cores exclusively to the requesting container, meaning no other container will have access to the core.

CPU Manager for Kubernetes performs a variety of operations to enable core pinning and isolation on a container or a thread level. These include:

- Discovering the CPU topology of the machine.
- Advertising the resources available via Kubernetes constructs.
- Placing workloads according to their requests.
- Keeping track of the current CPU allocations of the pods, ensuring that an application will receive the requested resources provided they are available.

CPU Manager for Kubernetes will create three distinct pools: exclusive, shared and infra. The exclusive pool is exclusive meaning only a single task may be allocated to a CPU at a time whereas the shared and infra pools are shared such that multiple processes may be allocated to a CPU. Following is an output for successful CPU Manager for Kubernetes deployment and CPU initialization. In the example setup, HT is enabled, therefore both physical and associated logical processors are isolated. Four cores are allocated for the CPU Manager for Kubernetes exclusive pool and 1 core for the shared pool. The rest of the CPU cores on the system are for the infra pool. The correct assignment of CPUs can be seen in the following example. We use node5 minion node as an example. You can change the node name to reflect your environment.

```
# kubectl logs cmk-init-install-discover-pod-node5 --namespace=kube-system init
INFO:root:Writing config to /etc/cmk.
INFO:root:Requested exclusive cores = 8.
INFO:root:Requested shared cores = 8.
INFO:root:Could not read SST-BF label from the node metadata: 'feature.node.kubernetes.io/cpu-
power.sst_bf.enabled'
```

```
INFO:root:Isolated logical cores:
16,17,18,19,20,21,22,23,40,41,42,43,44,45,46,47,64,65,66,67,68,69,70,71,88,89,90,91,92,93,94,95
INFO:root:Isolated physical cores: 16,17,18,19,20,21,22,23,40,41,42,43,44,45,46,47
INFO:root:Adding exclusive pool.
INFO:root:Adding cpu list 16,64 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 17,65 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 18,66 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 19,67 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 20,68 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 21,69 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 22,70 from socket 0 to exclusive pool.
INFO:root:Adding cpu list 23,71 from socket 0 to exclusive pool.
INFO:root:Adding shared pool.
INFO:root:Adding cpu list 40,88,41,89,42,90,43,91,44,92,45,93,46,94,47,95 to shared pool.
INFO:root:Adding infra pool.
INFO:root:Adding cpu list
0,48,1,49,2,50,3,51,4,52,5,53,6,54,7,55,8,56,9,57,10,58,11,59,12,60,13,61,14,62,15,63 to infra
pool.
INFO:root:Adding cpu list
24,72,25,73,26,74,27,75,28,76,29,77,30,78,31,79,32,80,33,81,34,82,35,83,36,84,37,85,38,86,39,87
to infra pool.
```

On successful run, the allocatable resource list for the node should be updated with resource discovered by the plugin as shown below. Note that the resource name like "cmk.intel.com/exclusive-cores".

```
# kubectl get node node5 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "8",
  "cpu": "95900m",
  "ephemeral-storage": "48294789041",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "8Gi",
  "intel.com/sriov_net": "32",
  "memory": "187760556Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

CPU Manager for Kubernetes ensures exclusivity, therefore the performance of latency sensitive workloads is not impacted by having a noisy neighbor on the system. CPU Manager for Kubernetes can be used along with the other Enhanced Platform Awareness (EPA) capabilities to achieve the improved network I/O, deterministic compute performance, and server platform sharing benefits offered by Intel® Xeon® Processor-based platforms.

## 7.4 Intel Device Plugins for Kubernetes

Like other vendors, Intel provides many hardware devices that deliver efficient acceleration of graphics, computation, data processing, security, and compression. Those devices optimize hardware for specific tasks, which saves CPU cycles for other workloads and typically results in performance gains. The Kubernetes device plugin framework provides a vendor-independent solution for hardware devices. Intel has developed a set of device plugins that comply with the Kubernetes device plugin framework and allow users to request and consume hardware devices across Kubernetes clusters such as Intel® QuickAssist Technology, GPUs, and FPGAs. The detailed documentation and code are available at:

- Documentation: https://builders.intel.com/docs/networkbuilders/intel-device-plugins-for-kubernetes-appnote.pdf
- Code: https://github.com/intel/intel-device-plugins-for-kubernetes

### 7.4.1 SR-IOV network device plugin

The Intel SR-IOV Network device plugin discovers and exposes SR-IOV network resources as consumable extended resources in Kubernetes. It works with SR-IOV VFs with both Kernel drivers and DPDK drivers. When a VF is attached with a kernel driver, then the SR-IOV CNI plugin can be used to configure this VF in the Pod. When using the DPDK driver, a VNF application configures this VF as required.

You need the SR-IOV CNI plugin for configuring VFs with the kernel driver in your pod. The DPDK driver supports VNFs that execute the VF driver and network protocol stack in userspace, allowing the application to achieve packet processing performance that greatly exceeds the ability of the kernel network stack.

By default, when VFs are created for Intel X710 NIC, these VFs are registered with the i40evf kernel module. VF with the Linux network driver uses the kernel network stack for all packet processing. To take advantage of user space packet processing with DPDK, a VF needs to be registered with either igb_uio or vfio-pci kernel module. On successful run, the allocatable resource list for

the node should be updated with resource discovered by the plugin as shown below. Note that the resource name is appended with the -resource-prefix, for example, "intel.com/sriov_net_A".

```
# kubectl get node node5 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "8",
  "cpu": "95900m",
  "ephemeral-storage": "48294789041",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "8Gi",
  "intel.com/sriov_net": "32",
  "memory": "187760556Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

### 7.4.2 Check Intel® QAT device plugin

Intel® QuickAssist adapters integrate hardware acceleration of compute intensive workloads such as bulk cryptography, public key exchange, and compression on Intel® Architecture Platforms. The Intel® QAT device plugin for Kubernetes supports Intel® QuickAssist adapters and includes an example scenario that uses the Data Plane Development Kit (DPDK) drivers. An additional demo that executes an Intel® QAT accelerated OpenSSL* workload with the Kata Containers runtime is also available. For more information, refer to Intel® QuickAssist adapters: https://www.intel.com/content/www/us/en/ethernetproducts/gigabit-server-adapters/quickassist-adapter-for-servers.html.

On successful run, the allocatable resource list for the node should be updated with resource discovered by the plugin as shown below. Note that the resource name uses a format similar to "qat.intel.com/generic".

```
# kubectl get node node5 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "8",
  "cpu": "95900m",
  "ephemeral-storage": "48294789041",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "8Gi",
  "intel.com/sriov_net": "32",
  "memory": "187760556Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

## 7.5 Check Advanced Networking Features

### 7.5.1 Multus CNI plugin

Kubernetes natively supports only a single network interface. Multus is a CNI plugin specifically designed to provide support for multiple networking interfaces in a Kubernetes environment. Operationally, Multus behaves as a broker and arbiter of other CNI plugins, meaning it invokes other CNI plugins (such as Flannel, Calico, SR-IOV, or userspace CNI) to do the actual work of creating the network interfaces. Multus v3.0 that has recently been integrated with Kubevirt, officially recognized as a CNCF project, and officially released with Kubespray v2.8.0. More information and source code can be found at: https://github.com/Intel-Corp/multus-cni

### 7.5.2 SR-IOV CNI plugin

Intel introduced the SR-IOV CNI plugin to allow a Kubernetes pod to be attached directly to an SR-IOV virtual function (VF) using the standard SR-IOV VF driver in the container host's kernel. Details on the SR-IOV CNI plugin v1.0.0 can be found at: https://github.com/intel/sriov-cni

1. Verify the networks using the following command:

```
# kubectl get net-attach-def
NAME            AGE
sriov-net       7d
userspace-ovs   7d
userspace-vpp   7d
```

### 7.5.3 Userspace CNI plugin

The Userspace CNI is a Container Network Interface plugin designed to implement user space networking such as DPDK based applications. The current implementation supports the DPDK-enhanced open vSwitch (OVS-DPDK) and Vector Packet Processing (VPP) along with the Multus CNI plugin in Kubernetes for the bare metal container deployment model. It enhances the high performance container networking solution and dataplane acceleration for NFV environment.

Verify the networks using the following command:

```
# kubectl get net-attach-def
NAME            AGE
sriov-net       7d
userspace-ovs   7d
userspace-vpp   7d
```

# 8 Use cases

This section provides several typical use cases to utilize the EPA features and advanced networking features of Kubernetes. The examples shown below are limited to one of two pods for simplicity, but can of course be scaled to a full solution as needed. The first example is a brief showcase of NFD, where the list of available nodes is filtered to only include those with SR-IOV configured.

Three types of network connectivity examples are shown: SR-IOV DPDK, SR-IOV CNI, and Userspace CNI. These showcase different methods for attaching pods to networks, depending on the need for inter- or intra-node connectivity. The last example shows how to use CPU Manager for Kubernetes to provide exclusive CPU core pinning to a pod, which can be useful for deterministic performance and behavior.

## 8.1 Use NFD to select the node to deploy the pod

With the help of NFD, the Kubernetes scheduler can now use the information contained in node labels to deploy pods according to the requirements identified in the pod specification. Using the nodeSelector option in the pod specification and the matching label on the node, a pod can be deployed on a SR-IOV configured node. If the scheduler does not find such a node in the cluster, then the creation of that pod will fail.

```
...
     nodeSelector:
       “node.alpha.kubernetes-incubator.io/nfd-network-SR-IOV-configured”: “true”
...
```

Together with other EPA technologies, including device plugins, NFD facilitates workload optimization through resource-aware scheduling. In particular, NFD can benefit workloads that utilize modern vector data processing instructions, require SR-IOV networking, and have specific kernel requirements.

## 8.2 Pod using SR-IOV

With the SR-IOV network device plugin and advanced network features for Kubernetes, a pod can take advantage of SR-IOV DPDK or SR-IOV CNI plugin to achieve the best network performance. The following examples are useful when connecting outside of a minion node (inter-node), because the interface is attached to the NIC via SR-IOV, either directly or through the CNI plugin.

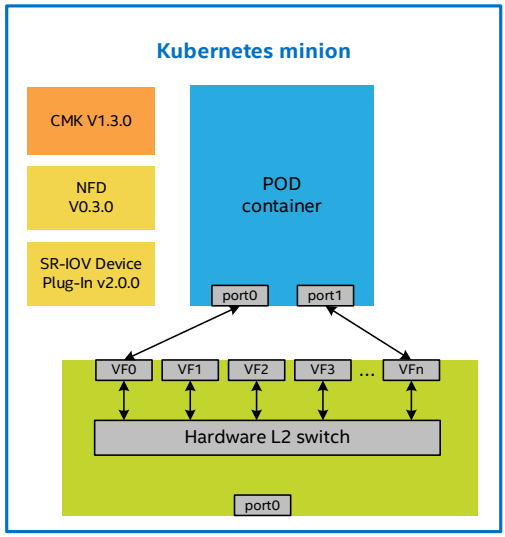
### 8.2.1 Pod using SR-IOV DPDK

1. Create the pod spec file sriov-dpdk-res.yaml.
   In the example below, two VFs are requested from the ‘sriov_net_b’ resource that was created previously using the SR-IOV network device plugin. This method assigns VFs to the pod, without providing any interface configuration or IPAM. This makes it ideal for DPDK applications, where the configuration is done directly in the application.

```
# cat sriov-dpdk-res.yaml
apiVersion: v1
kind: Pod
metadata:
  name: sriov-dpdk-pod
  labels:
    env: test
spec:
  tolerations:
  - operator: "Exists"
  containers:
  - name: appcntr1
    image: centos/tools
    imagePullPolicy: IfNotPresent
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        intel.com/sriov_net_B: '2'
      limits:
        intel.com/sriov_net_B: '2'
```

2. Create the pod from the master node using sriov-dpdk-res.yaml:

```
# kubectl create -f sriov-dpdk-res.yaml
pod/sriov-dpdk-pod created
```




3. Inspect the pod environment, you can find the VFs PCI address from the environment variables, shown below using the env command. The PCI address of the VFs used by the pod are 0000:18:10.3 and 0000:18:10.5 in this example.

```
# kubectl exec -ti sriov-dpdk-pod env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=sriov-dpdk-pod
TERM=xterm
PCIDEVICE_INTEL_COM_SRIOV_NET_B=0000:18:10.3,0000:18:10.5
KUBERNETES_PORT=tcp://10.233.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.233.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.233.0.1
KUBERNETES_SERVICE_HOST=10.233.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
container=docker
HOME=/root
```

At this point, the application running in the pod can decide what to do with the VFs.

### 8.2.2 Pod using SR-IOV CNI plugin

1. Create a sample pod specification sriov-test.yaml file.
   In this example, two different networks objects are used, 'flannel-networkobj' for access and management, and 'sriov-net' for the SR-IOV interface which is already configured using the IPAM settings provided for the object.

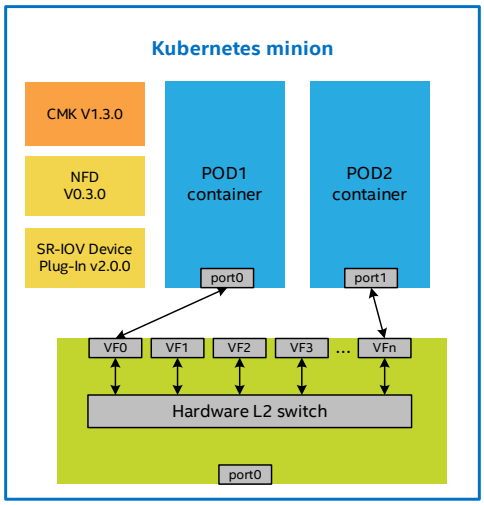
```
# cat sriov-test.yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: sriov-pod-
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "flannel-networkobj" },
            { "name": "sriov-net" }
    ]'
spec:  # specification of the pod's contents
  tolerations:
  - operator: "Exists"
  containers:
  - name: multus-multi-net-poc
    image: busybox
    command: ["top"]
    stdin: true
    tty: true
    resources:
      requests:
        intel.com/sriov_net_A: '1'
      limits:
        intel.com/sriov_net_A: '1'
```

2. Create two multiple network-based pods from the master node using sriov-test.yaml:

```
# kubectl create -f sriov-test.yaml -f sriov-test.yaml
pod/sriov-pod-x7g49 created
pod/sriov-pod-qk24t created
```

3. Retrieve the details of the running pod from the master:

```
# kubectl get pods
NAME                  READY     STATUS    RESTARTS   AGE
sriov-pod-qk24t       1/1       Running   0          5h1m
sriov-pod-x7g49       1/1       Running   0          5h1m
```




Once the configuration is complete, you can verify the pod networks are working as expected. To verify these configurations, complete the following steps:

1. Run "ifconfig" command inside the container:

```
# kubectl exec sriov-pod-x7g49 -ti -- ifconfig
eth0      Link encap:Ethernet  HWaddr 0A:58:0A:E9:44:12
          inet addr:10.233.68.18  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:13 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:983 (983.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          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:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

net1      Link encap:Ethernet  HWaddr 0A:58:0A:E9:44:02
          inet addr:10.233.68.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:13 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:983 (983.0 B)  TX bytes:0 (0.0 B)

net2      Link encap:Ethernet  HWaddr 72:DD:4B:F6:C1:30
          inet addr:192.168.1.40  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1772025 errors:0 dropped:0 overruns:0 frame:0
          TX packets:662172 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:567745190 (541.4 MiB)  TX bytes:110196136 (105.0 MiB)

# kubectl exec sriov-pod-qk24t -ti -- ifconfig
eth0      Link encap:Ethernet  HWaddr 0A:58:0A:E9:43:0F
          inet addr:10.233.67.15  Bcast:0.0.0.0  Mask:255.255.255.0
```

```
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:12 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:824 (824.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          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:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

net1      Link encap:Ethernet  HWaddr 0A:58:0A:E9:43:04
          inet addr:10.233.67.4  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:12 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:824 (824.0 B)  TX bytes:0 (0.0 B)

net2      Link encap:Ethernet  HWaddr 5A:33:6A:C5:B9:7D
          inet addr:192.168.1.42  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1937001 errors:0 dropped:0 overruns:0 frame:0
          TX packets:756208 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:634654278 (605.2 MiB)  TX bytes:125495488 (119.6 MiB)
```

In the above output, 'net1' is the flannel interface (each pod is on a separate subnet), and 'net2' is the SR-IOV interface (shared subnet). This provides separation of the external access, while still allowing pods to communicate on a shared subnet.

2. Verify the networks by pinging from one pod to another through the SR-IOV (net2) interface:

```
# kubectl exec sriov-pod-x7g49 -ti -- ping 192.168.1.42
PING 192.168.1.40 (192.168.1.40): 56 data bytes
64 bytes from 192.168.1.42: seq=0 ttl=64 time=0.131 ms
64 bytes from 192.168.1.42: seq=1 ttl=64 time=0.102 ms
64 bytes from 192.168.1.42: seq=2 ttl=64 time=0.124 ms
64 bytes from 192.168.1.42: seq=3 ttl=64 time=0.113 ms
64 bytes from 192.168.1.42: seq=4 ttl=64 time=0.106 ms
64 bytes from 192.168.1.42: seq=5 ttl=64 time=0.108 ms
```

## 8.3 Pod using Userspace CNI plugin

The Userspace CNI plugin example focuses on connectivity between pods mainly residing on the same node (intra-node), which is provided through the Userspace CNI that connects the interface to OVS or VPP using vhost-user.

### 8.3.1 Pod using Userspace CNI with OVS-DPDK

The following example creates a testpmd pod with two interfaces using the 'userspace-networkobj' object, which creates vhost-user interfaces that are connected to OVS. Since this is using vhost-user, the pod is also configured with a volume mount for the socket file, and a huge page memory resource.

***Note:*** 1G hugepages are not available by default and need to be created using kernel boot parameters before the system boots up. Refer to Section 5.3 for details.

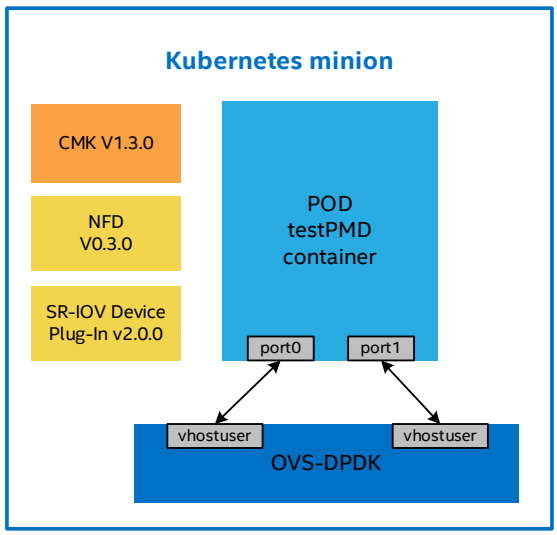
Here is the testpmd pod specification example file.

```
# cat testpmd.yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: testpmd-
  annotations:
    k8s.v1.cni.cncf.io/networks: userspace-networkobj, userspace-networkobj
spec:
  containers:
  - name: testpmd
    image: testpmd:v1.0
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
```

```
      runAsUser: 0
    volumeMounts:
    - mountPath: /vhu/
      name: socket
    - mountPath: /dev/hugepages1G
      name: hugepage
    resources:
      requests:
        memory: 1Gi
      limits:
        hugepages-1Gi: 1Gi
    command: ["sleep", "infinity"]
  tolerations:
  - operator: "Exists"
  volumes:
  - name: socket
    hostPath:
      path: /var/lib/cni/vhostuser/
  - name: hugepage
    emptyDir:
      medium: HugePages
  securityContext:
    runAsUser: 0
  restartPolicy: Never
```

1. Create the testpod pod from the master node:

```
# kubectl create -f testpmd.yaml
pod/testpmd-wfw9v created
```




2. Run this command to enter the testpmd container:

```
# kubectl exec -ti testpmd-wfw9v bash
```

3. Inside the testpmd container, run these commands:

```
root@ testpmd-wfw9v:/# export ID=$(/vhu/get-prefix.sh)
root@multi-vhost-examplewfw9v:/# testpmd -d librte_pmd_virtio.so.17.11 -m 1024 -c 0xC \
  --file-prefix=testpmd_ --vdev=net_virtio_user0,path=/vhu/${ID}/${ID:0:12}-net1 \
  --vdev=net_virtio_user1,path=/vhu/${ID}/${ID:0:12}-net2 --no-pci -- --no-lsc-interrupt \
  --auto-start --tx-first --stats-period 1 --disable-hw-vlan
```

You will see output similar to the following screenshot, which means your testpmd app can send and receive data packages. Both ports are connected through the NIC, and connectivity is verified by packets being both sent and received on both ports.

```
Port statistics ================================
  ######################## NIC statistics for port 0  ########################
  RX-packets: 2111001504 RX-missed: 0          RX-bytes:  135104096256
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 2105280704 TX-errors: 0          TX-bytes:  134737965056

  Throughput (since last show)
  Rx-pps:      1397336
  Tx-pps:      1392591
  ############################################################################

  ######################## NIC statistics for port 1  ########################
  RX-packets: 2105280768 RX-missed: 0          RX-bytes:  134737969152
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 2111001600 TX-errors: 0          TX-bytes:  135104102400

  Throughput (since last show)
  Rx-pps:      1392591
  Tx-pps:      1397336
  ############################################################################
```

You also can have two or more pods connect the OVS via Userspace CNI plugin. Here is an example using two pods, one is pktgen and one is l3fwd.

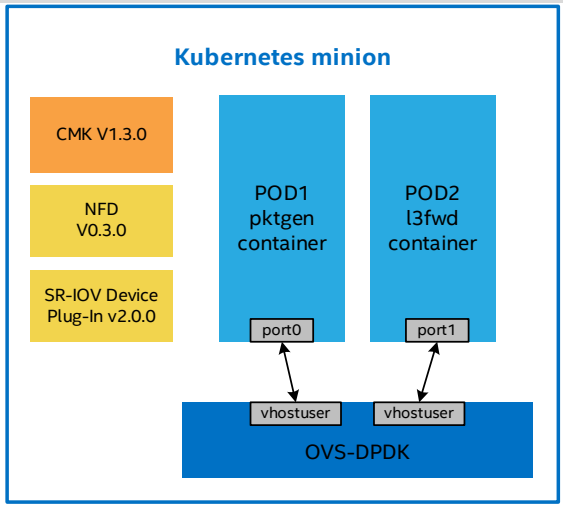
1. Create a pktgen pod specification pktgen.yaml.
   This is similar to the testpmd.yaml specification from above, but instead of having two interfaces this only has one.

```
# cat pktgen.yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: dpdk-pktgen-
  annotations:
    k8s.v1.cni.cncf.io/networks: userspace-networkobj
spec:
  containers:
  - name: dpdk-pktgen
    image: dpdk-pktgen:v1.0
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
      runAsUser: 0
    volumeMounts:
    - mountPath: /vhu/
      name: socket
    - mountPath: /dev/hugepages1G
      name: hugepage
    resources:
      requests:
        memory: 2Gi
      limits:
        hugepages-1Gi: 2Gi
    command: ["sleep", "infinity"]
  tolerations:
  - operator: "Exists"
  nodeSelector:
    kubernetes.io/hostname: node5
  volumes:
  - name: socket
    hostPath:
      path: /var/lib/cni/vhostuser/
  - name: hugepage
    emptyDir:
      medium: HugePages
  securityContext:
    runAsUser: 0
  restartPolicy: Never
```

2. Create l3fwd pod specification l3fwd.yaml, which uses similar resources as pktgen.yaml, but with a different name and image:

```
# cat l3fwd.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  generateName: dpdk-l3fwd-
  annotations:
    k8s.v1.cni.cncf.io/networks: userspace-networkobj
spec:
  containers:
  - name: dpdk-l3fwd
    image: dpdk-l3fwd:v1.0
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
      runAsUser: 0
    volumeMounts:
    - mountPath: /vhu/
      name: socket
    - mountPath: /dev/hugepages1G
      name: hugepage
    resources:
      requests:
        memory: 1Gi
      limits:
        hugepages-1Gi: 1Gi
    command: ["sleep", "infinity"]
  tolerations:
  - operator: "Exists"
  nodeSelector:
    kubernetes.io/hostname: node5
  volumes:
  - name: socket
    hostPath:
      path: /var/lib/cni/vhostuser/
  - name: hugepage
    emptyDir:
      medium: HugePages
  securityContext:
    runAsUser: 0
  restartPolicy: Never
```




3. Create the pktgen pod from the master node:

```
# kubectl create -f pktgen.yaml
pod/dpdk-pktgen-knpp4 created
```

4. Run this command to enter the pktgen container from master mode:

```
# kubectl exec -ti dpdk-pktgen-knpp4 bash
```

5. Run the following commands inside the pktgen container:

```
root@dpdk-pktgen-knpp4:/usr/src/pktgen# export ID=$(/vhu/get-prefix.sh)
root@dpdk-pktgen-knpp4:/usr/src/pktgen# ./app/x86_64-native-linuxapp-gcc/pktgen -l 10,11,12 --
vdev=virtio_user0,path=/vhu/${ID}/${ID:0:12}-net1 --no-pci --socket-mem=512 --master-lcore 10 -
- -m 11:12.0 -P
Pktgen:/>
```

The pktgen application should launch. Among the statistics, make note of the pktgen source MAC address listed as 'Src MAC Address'.

6. In another terminal, create the l3fwd pod from the master node:
```
# kubectl create -f l3fwd.yaml
pod/dpdk-l3fwd-t4ppr created
```
7. Run this command to enter the l3fwd container from master node:
```
# kubectl exec -ti dpdk-l3fwd-t4ppr bash
```
8. Inside the l3fwd container, export the port ID prefix and start the l3fwd application. Set the destination MAC address using the 'dest' argument. This should be the Src MAC Address previously noted from the pktgen pod (adjust cores, memory, etc. to suit your system):
```
root@dpdk-l3fwd-t4ppr:/usr/src/dpdk/examples/l3fwd/x86_64-native-linuxapp-gcc/app# export
ID=$(/vhu/get-prefix.sh)
root@dpdk-l3fwd-t4ppr:/usr/src/dpdk/examples/l3fwd/x86_64-native-linuxapp-gcc/app# ./l3fwd -c
0x10 --vdev=virtio_user0,path=/vhu/${ID}/${ID:0:12}-net1 --no-pci --socket-mem=512 -- -p 0x1 -P
--config "(0,0,4)" --eth-dest=0, ,<pktgen-source-mac-add> --parse-ptype
```
The l3fwd app should start up. Among the information printed to the screen will be the 'Address'. This is the MAC address of the l3fwd port, make note of it.
9. Back on the pktgen pod, set the destination MAC address to that of the l3fwd port:
```
Pktgen:/> set 0 dst mac <l3fwd-mac-address>
```
10. Start traffic generation:
```
Pktgen:/> start 0
```
You should see the packet counts for Tx and Rx increase, verifying that packets are being transmitted by pktgen and are being sent back via l3fwd running in the other pod.
11. To exit:
```
Pktgen:/> stop 0
Pktgen:/> quit
```

```
- Ports 0-0 of 1   <Main Page>  Copyright (c) <2010-2017>, Intel Corporation
  Flags:Port      :   P--------------:0
Link State        :       <UP-10000-FD>     ----TotalRate----
Pkts/s Max/Rx     :     6314816/6156640       6314816/6156640
      Max/Tx      :     6728512/6713792       6728512/6713792
MBits/s Rx/Tx     :           4137/4511             4137/4511
Broadcast         :                   0
Multicast         :                   0
  64 Bytes        :           249869184
  65-127          :                   0
  128-255         :                   0
  256-511         :                   0
  512-1023        :                   0
  1024-1518       :                   0
Runts/Jumbos      :                 0/0
Errors Rx/Tx      :                 0/0
Total Rx Pkts     :           249080800
      Tx Pkts     :           266350944
      Rx MBs      :              167382
      Tx MBs      :              178987
ARP/ICMP Pkts     :                 0/0
                  :
Pattern Type      :                abcd...
Tx Count/% Rate   :        Forever /100%
PktSize/Tx Burst  :             64 /   64
Src/Dest Port     :          1234 / 5678
Pkt Type:VLAN ID  :      IPv4 / TCP:0001
802.1p CoS        :                   0
ToS Value:        :                   0
  - DSCP value    :                   0
  - IPP  value    :                   0
Dst  IP Address   :          192.168.1.1
Src  IP Address   :        192.168.0.1/24
Dst MAC Address   :    c6:53:7c:29:8e:f4
Src MAC Address   :    f6:0d:f2:66:bc:9c
VendID/PCI Addr   :    0000:0000/00:00.0

-- Pktgen Ver: 3.4.8 (DPDK 17.11.3)  Powered by DPDK -------------------------

** Version: DPDK 17.11.3, Command Line Interface with timers
Pktgen:/> set 0 dst mac C6:53:7C:29:8E:F4
Pktgen:/> start 0
Pktgen:/>
```

You also can use Userspace CNI with VPP; it is similar to OVS-DPDK. Learn more at:
https://builders.intel.com/docs/networkbuilders/adv-network-features-in-kubernetes-app-note.pdf

## 8.4 Pod using CPU Pinning and Isolation in Kubernetes

To run testpmd using CPU Pinning and Isolation in Kubernetes, perform the steps below.

1. Add the following in the testpmd pod spec in previous testpmd.yaml file. You can change the core number you want to use for the pod. In following case, we just use 4 exclusive cores in this example.

```
  resources:
      requests:
        cmk.intel.com/exclusive-cores: '4'
      limits:
        cmk.intel.com/exclusive-cores: '4'
```

   ***Note:*** If the webhook is used, then you need an additional annotation. If the webhook is not used, you need additional mount points.

   For more details, refer to a sample pod spec at: https://github.com/intel/CPU-Manager-for-Kubernetes/blob/master/resources/pods/cmk-isolate-pod.yaml

2. Deploy testpmd pod and connect to it using a terminal window:

```
# kubectl create -f testpmd-cmk.yaml
pod/testpmd-cmk-pm3uu created
# kubectl exec testpmd-cmk-pm3uu -ti bash
```

3. Create /etc/cmk/use_cores.sh file with the following content:

```
#!/bin/bash
export CORES=`printenv CMK_CPUS_ASSIGNED`
SUB=${ID:0:12}
COMMAND=${@//'$CORES'/$CORES}
COMMAND=${COMMAND//'$ID'/$ID}
COMMAND=${COMMAND//'$SUB'/$SUB}
$COMMAND
```

   ***Note:*** The above script uses CMK to assign the cores from temporary environment variable 'CMK_CPUS_ASSIGNED' to its local variable CORES. Then, this variable substitutes the $CORES phrase in the command provided below as argument to this script and executes it with the correct cores selected.

4. Add executable rights to the script:

```
# chmod +x /etc/kcm/use_cores.sh
```

5. Start testpmd using use_cores.sh script:

```
root@ testpmd-cmk-pm3uu:/# export ID=$(/vhu/get-prefix.sh)
root@multi-vhost-examplewfw9v:/# /opt/bin/cmk isolate --conf-dir=/etc/cmk --pool=exclusive --
no-affinity /etc/cmk/use_cores.sh 'testpmd -d librte_pmd_virtio.so.17.11 -m 1024 -l \$CORES --
file-prefix=testpmd_ --vdev=net_virtio_user0,path=/vhu/\$ID/\$SUB-net1 --
vdev=net_virtio_user1,path=/vhu/\$ID/\$SUB-net2 --no-pci -- --no-lsc-interrupt --auto-start --
tx-first --stats-period 1 --disable-hw-vlan'
```

The testpmd has requested exclusive cores from CPU Manager for Kubernetes, which have been advertised to the workload via environment variables. The workload is using the –no-affinity option, which indicates that CPU Manager for Kubernetes has left the pinning of the cores to the application and is pinning the CPU Manager for Kubernetes assigned cores itself using the variable from the script. The testpmd can get deterministic performance due to the guaranteed CPUs exclusivity by CPU Manager for Kubernetes.

CPU Manager for Kubernetes can be utilized along with the other Enhanced Platform Awareness (EPA) capabilities to achieve the improved network I/O, deterministic compute performance, and server platform sharing benefits offered by Intel® Xeon® Processor-based platforms.

CPU Pinning and Isolation capability is part of EPA methodology and the related suite of changes across the orchestration layers' stack. EPA methodology enables platform capabilities discovery, intelligent configuration and workload-placement decisions resulting in improved and deterministic application performance.

# 9 Conclusion

Intel and its partners have been working with open source communities to add new techniques and address key barriers to adoption for commercial containers by harnessing the power of Intel Architecture-based servers to improve configuration, manageability, deterministic performance, network throughput, service-assurance and resilience of container deployments.

This document contains brief, high-level notes on installation, configuration, and use of Enhanced Platform Awareness (EPA), networking, and device plugin features for Kubernetes. By following this document, it is possible to set up a Kubernetes cluster and add simple configurations for some of the features provided by Intel. The included example usage cases show how the features can be consumed to provide additional functionality in both Kubernetes and the deployed pods, including but not limited to flexible network configurations, Node Feature Discovery, and CPU pinning for exclusive access to host cores.

We highly recommend that you take advantage of these EPA features, advanced network features and device plugins in container-based NFV deployment.



0619/DN/PTI/PDF

338934-002US