REFERENCE ARCHITECTURE RELEASE V1.8

Intel Corporation

Container Bare Metal for 2nd Generation Intel® Xeon® Scalable Processor Reference Architecture

Authors

Sreemanti Ghosh

Abdul Halim

Przemyslaw Lal

Gary Loughnane

David Lu

Killian Muldoon

Dana Nehama

Michael Pedersen

Monika Rymsza

1 Introduction

This guide is a Reference Architecture document that provides instructions on how to build a Bare Metal Container setup on an Intel platform using accelerators. The goal of this guide is to help developers adopt and use advanced networking technologies and device plugin features in a container bare metal reference architecture based on 2nd generation Intel® Xeon® Scalable processors and other Intel technologies.

This document describes the set of combined hardware and software components forming the reference architecture with emphasis on the configuration, installation, and use of networking and device plugin features for Kubernetes*. The document also describes a configuration and installation playbook, which enables users to perform automated deployments, thus decreasing installation time from days to hours.

This version of the Reference Architecture (Release v1.8) has been updated to include support for the following technologies:

- Bond Container Networking Interface (CNI)
- Telemetry Aware Scheduling
- Topology Manager
- Dynamic Device Personalization (DDP)

This document is part of the Container Experience Kit, which is available at: https://networkbuilders.intel.com/network-technologies/container-experience-kits

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	April 2019	Initial release of document.
002	June 2019	Added technical details for using Ansible playbook. Updates to document formatting and layout.
003	September 2019	Enhanced Ansible playbook description and instructions. Updates to support Kubernetes v1.13.5, Docker v18.6.2, and DPDK v18.11.1.
004	November 2019	Updated SR-IOV Network device plugin version to v3.0.0. Added automation for driver binding. Updated Ansible version to 2.7.12.
005	February 2020	Added support for Kubernetes v1.14, v1.15, v1.16 and dedicated software BOM tables. Added Topology Manager enablement and validation steps. Updated example configurations and variables descriptions. Updated installation steps (added shell command to checkout desired version). Updated authors and acronyms table.
006	April 2020	Updated DPDK version to 19.11 and OVS-DPDK 2.13.0. Added Base Frequency configuration. Added support for Bond Container Networking Interface (CNI), Telemetry Aware Scheduling, Topology Manager, and Dynamic Device Personalization (DDP). Updated hugepages default value to 1G. Updated authors and acronyms table. Removed Kubernetes 1.13.

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
BIOS	Basic Input / Output System
BMRA	Bare Metal Reference Architecture
CMK	Intel® CPU Manager for Kubernetes
CNI	Container Networking Interface
DDP	Dynamic Device Personalization
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® SST-BF	Intel® Speed Select Technology – Base Frequency
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
OVS DPDK	Open vSwitch with DPDK
QAT	QuickAssist Technology
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
TAS	Telemetry Aware Scheduling
vBNG	Virtual Broadband Network Gateway
vCMTS	Virtual Cable Modem Termination System
VLAN	Virtual LAN
VNF	Virtual Network Function
VPP	Vector Packet Processing
VXLAN	Virtual Extensible LAN

1.2 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 Management - CPU Pinning and Isolation in Kubernetes* Technology Guide	https://builders.intel.com/docs/networkbuilders/cpu-pin-and-isolation-in-kubernetes-app-note.pdf

REFERENCE	SOURCE
Intel Device Plugins for Kubernetes Application Note	https://builders.intel.com/docs/networkbuilders/intel-device-plugins-for-kubernetes-appnote.pdf
Intel® Speed Select Technology – Base Frequency - Enhancing Performance Application Note	https://builders.intel.com/docs/networkbuilders/intel-speed-select-technology-base-frequency-enhancing-performance.pdf
Topology Management - Implementation in Kubernetes* Technology Guide	https://builders.intel.com/docs/networkbuilders/topology-management-implementation-in-kubernetes-technology-guide.pdf
Telemetry Aware Scheduling - Automated Workload Optimization with Kubernetes* (K8s*) Technology Guide	https://builders.intel.com/docs/networkbuilders/telemetry-aware-scheduling-automated-workload-optimization-with-kubernetes-k8s-technology-guide.pdf
Intel® Ethernet Controller 700 Series - Dynamic Device Personalization Support for CNF with Kubernetes* Technology Guide	https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-dynamic-device-personalization-support-for-cnf-with-kubernetes-technology-guide.pdf

2 Technology Overview

Containers are increasingly important in cloud computing and fundamental to Cloud Native adoption. A container is lightweight, agile and portable, and it can be quickly created, updated, and removed. Kubernetes* (K8s*) 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 a suite of capabilities and methodologies that exposes Intel® Architecture platform features for increased and deterministic application and network performance.

The technology described in this document consists of capabilities implemented across the software stack, which enables a Kubernetes environment that leverages Intel technologies. It targets intelligent platform capability, configuration, and capacity data consumption. Intel and partners have worked together to advance the discovery, scheduling, and isolation of server hardware features using the following technologies:

- Dynamic Device Personalization (DDP) is one of the key technologies of the Intel® Ethernet 700 Series. It enables workload-specific optimizations using the programmable packet processing pipeline to support a broader range of traffic types.
- Node Feature Discovery (NFD) enables generic hardware capability discovery in Kubernetes, including Intel® Xeon® processor-based hardware.
- Bond CNI allows for aggregating multiple network interfaces into one logical interface.
- Telemetry Aware Scheduling (TAS) is a Kubernetes add-on that consumes cluster metrics and makes intelligent scheduling decisions based on operator-defined policies. It enhances automation and enables optimization of resource management at the cluster level.
- Native CPU Manager and Intel CPU Manager for Kubernetes provide mechanisms for CPU core pinning and isolation of containerized workloads.
- Topology Manager, a native component of Kubernetes v1.16, enables other Kubelet components to make resource allocation decisions using topology-based information.
- Hugepages support, added to Kubernetes v1.8, enables the discovery, scheduling and allocation of hugepages as a native first-class resource. This support addresses low latency and deterministic memory access requirements.
- Single Root Input/Output Virtualization (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.
- Device plugins, including GPU, FPGA, Intel® QuickAssist Technology, and SR-IOV device plugins boost performance and platform efficiency. A set of Ansible* and Helm* scripts which enable easy and fast automatic installation on a container bare metal NFV platform. Refer to Section 3.2 for details.

One of the important features included is Multus, which enables support of 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.

Other networking features introduced are the SR-IOV CNI plugin and the 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 an SR-IOV Virtual Function (VF) using the standard SR-IOV VF driver in the container host's kernel. The Userspace CNI plugin is 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 data plane acceleration in NFV environments.

Intel delivers a set of device plugins that provide access to technologies that free up CPU cycles and boost performance. The plugins deliver efficient acceleration of graphics, compute, data processing, security, and compression and include the following:

- **GPU device plugin:** VNF/CNFs 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.

3 Physical and Software Topology

This section describes the topology of the reference architecture.

3.1 Physical Topology

This section describes the physical topology for the Intel Container Bare Metal Reference Architecture, which uses 3 master nodes and 2 worker nodes in a cluster setup. The diagram also shows the advanced network features and technologies that were used, including CPU Manager for Kubernetes, Node Feature Discovery (NFD), SR-IOV Network Device Plugin, and others.

This Reference Architecture uses testpmd and pktgen pods and sample VNF workloads (vCMTS and vBNG) for integration validation. Figure 1 shows the physical topology for the reference architecture described in this document.

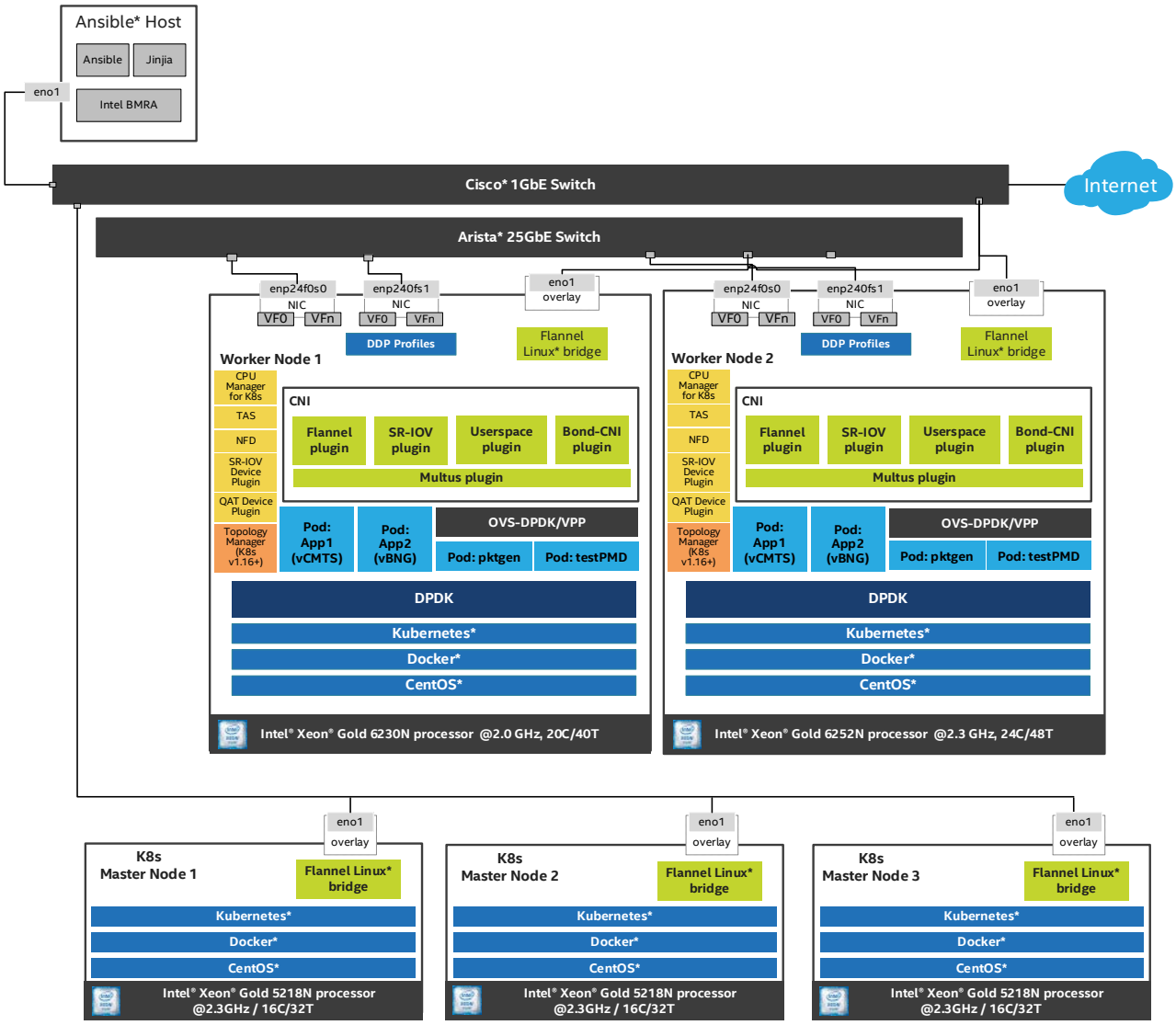


Figure 1. Reference Architecture Topology[1]

[1] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

3.2 Container Bare Metal Reference Architecture (BMRA) Installation Playbook

Intel provides a set of Ansible* and Helm* scripts which enable easy and fast automatic installation on a container bare metal NFV platform. Ansible is an agentless configuration management tool that uses playbooks to perform actions on many machines and Helm is a package manager tool which runs on top of Kubernetes to automate the installation process of plugins and K8s capabilities. The playbook takes into consideration the Intel BKC (Best Known Configuration) for optimized performance. The installation is done using the master Ansible playbook which performs initial system configuration, deploys Kubernetes and its add-ons via Kubespray followed by the K8s capabilities. The playbook enables users to customize multiple parameters to fit their installation requirements.

Figure 2 shows the three high-level Ansible playbook components, which include:
1. Infrastructure setup block
2. Kubernetes setup block
3. Capabilities setup block

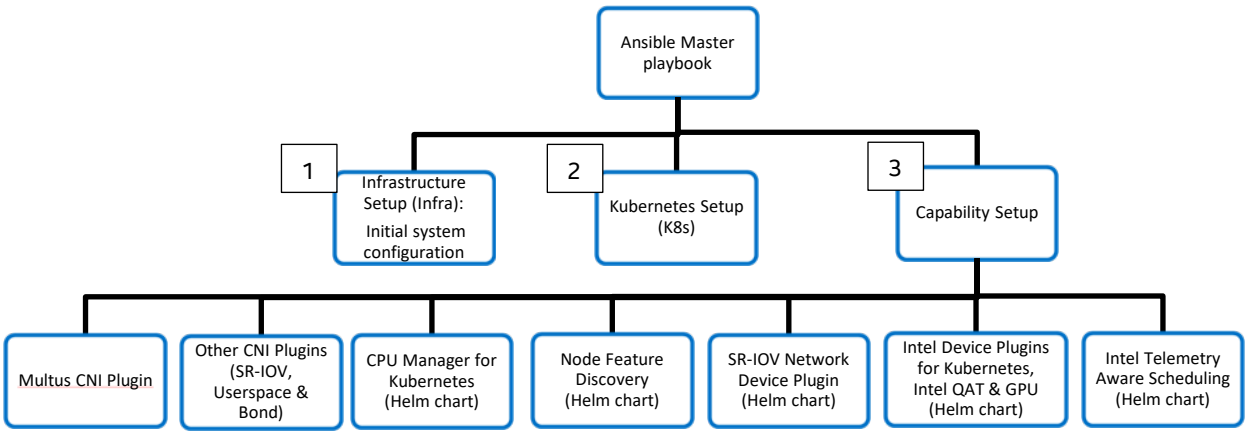


Figure 2. BMRA High Level Ansible Playbook

Figure 3 describes the detailed Ansible playbook components for software installation described in this document.

Ansible requires the following components to work: Inventory files, group_vars, host_vars, playbooks, roles and tasks. You can customize multiple parameters in these components to fit your installation requirements. Refer to Section 7 for details.

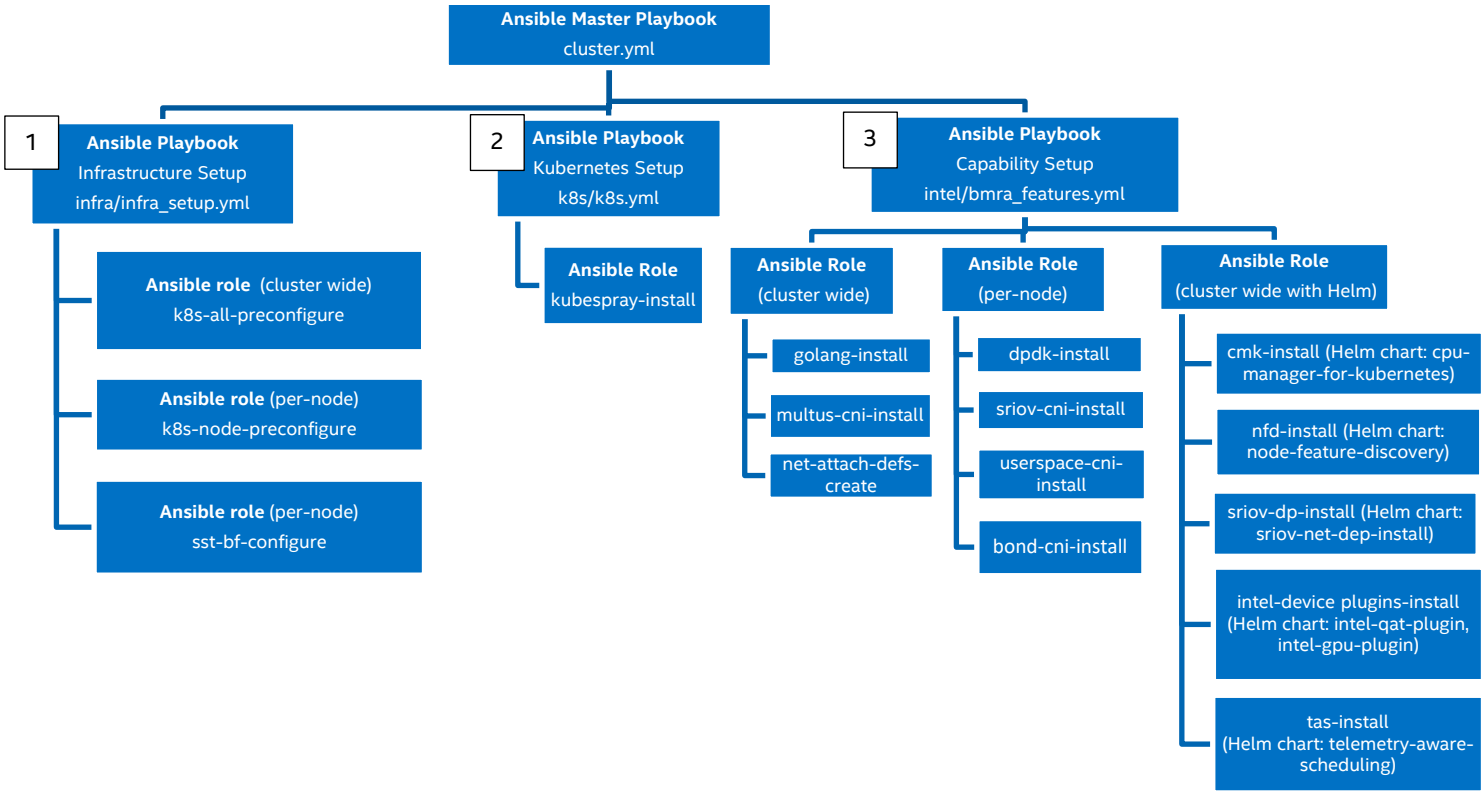


Figure 3. BMRA Detailed Ansible Playbook

The Ansible playbook runs three sub playbooks, as shown in Figure 3. The 3 sub playbooks in order of their installation are:

Infrastructure Set Up (named infra playbook): This playbook modifies kernel boot parameters and sets the initial system configuration in the cluster.

- Uses the 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.
- User can set the size and number of Hugepages as per requirement. The default size is set to 1G which can be reconfigurable.
- SR-IOV virtual functions are created by writing an appropriate value to the `sriov_numvfs` parameter. An appropriate SR-IOV PCI address must be used to fit the environment.
- SST-BF configuration is enabled, which provides control over base frequency.

Kubernetes Set Up (named k8s playbook): This playbook deploys a High Availability (HA) K8s cluster using Kubespray, which is a project under the Kubernetes community that deploys production-ready Kubernetes clusters. The Multus CNI plugin, which is specifically designed to provide support for multiple networking interfaces in a K8s environment, is deployed by Kubespray along with Flannel, Docker registry, and Helm server.

Intel BM RA Capability Setup (named Intel playbook): Advanced networking technologies, Enhanced Platform Awareness, and device plugin features are deployed by this playbook using Helm Charts as part of the BM RA. The following capabilities are deployed:

- Multus CNI plugin: Though Multus is deployed as part of Kubespray in the K8s playbook, this gives the user the additional benefit to use a different version of Multus to the one supported by Kubespray and also allows the user the flexibility to install only the Intel capabilities on top of an existing K8s cluster if the K8s playbook is not required.
- SR-IOV CNI plugin, which allows a Kubernetes pod to be attached directly to an SR-IOV VF using the standard SR-IOV VF driver in the container host's kernel.
- Userspace CNI plugin, which is a Container Network Interface plugin designed to implement user space networking, such as DPDK based applications.
- Bond CNI plugin, which allows for aggregating multiple network interfaces into a single logical interface.

- CPU Manager for Kubernetes, which performs a variety of operations to enable core pinning and isolation on a container or a thread level.
- Node Feature Discovery (NFD), which is a K8s add-on to detect and advertise hardware and software capabilities of a platform that can, in turn, be used to facilitate intelligent scheduling of a workload.
- SR-IOV network device plugin, which discovers and exposes SR-IOV network resources as consumable extended resources in Kubernetes.
- Intel® QAT device plugin, in which a workload can alleviate pipeline bottlenecks and improve performance by offloading cryptographic operations of user traffic to an Intel® QAT device.
- GPU device plugin, which offloads the processing of computation intensive workloads to GPU hardware.
- Telemetry Aware Scheduling, which allows for scheduling workloads based on telemetry data.

4 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 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 Each NIC assigned to a separate NUMA node (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 Intel® C627/C628 Chipset	Integrated w/NUMA connectivity to each CPU or minimum 16 Peripheral Component Interconnect express* (PCIe*) lane Connectivity to one CPU
BIOS	Intel Corporation SE5C620.86 B.0D.01.0241 Release Date: 11/19/2018	Intel® Hyper-Threading Technology (Intel® HT Technology) enabled Intel® Virtualization Technology (Intel® VT-x) enabled Intel® Virtualization Technology for Directed I/O (Intel® VT-d) enabled
Switches	Cisco* Catalyst 2960-XR Arista* DCS-7280QR-C36-R	Cisco 1GbE Switch Arista 25GbE Switch

5 Software BOM

This section describes the software version details for each supported Kubernetes release (v1.14, v1.15, and v1.16). It also includes Table 7, which shows the software versions used for each Kubernetes release.

5.1 Software BOM using Kubernetes v1.14

Table 4. Software BOM using Kubernetes v1.14

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Host OS	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64	https://www.centos.org/
Ansible*	Ansible v2.7.16	https://www.ansible.com/
BMRA Ansible Playbook	v1.2.1	https://github.com/intel/container-experience-kits
Python*	Python 2.7	https://www.python.org/
Kubespray*	Kubespray v2.10	https://github.com/kubernetes-sigs/kubespray
Docker*	Docker 19.03.8	https://www.docker.com/
Container orchestration engine	Kubernetes v1.14.6	https://github.com/kubernetes/kubernetes
CPU Manager for Kubernetes*	CPU Manager for Kubernetes v1.4.0	https://github.com/intel/CPU-Manager-for-Kubernetes
Telemetry Aware Scheduling	Telemetry Aware Scheduling	https://github.com/intel/telemetry-aware-scheduling
Node Feature Discovery	NFD v0.3.0	https://github.com/kubernetes-sigs/node-feature-discovery
Data Plane Development Kit	DPDK 19.11	https://core.dpdk.org/download/
Open vSwitch with DPDK	OVS-DPDK v2.13.0	http://docs.openvswitch.org/en/latest/intro/install/dpdk/
Vector Packet Processing	VPP 18.07	https://docs.fd.io/vpp/18.07/index.html
Multus CNI	Multus CNI v3.2	https://github.com/intel/multus-cni
SR-IOV CNI	SR-IOV CNI v2.0.0	https://github.com/intel/sriov-cni
SR-IOV network device plugin	SR-IOV network device plugin v3.1	https://github.com/intel/sriov-network-device-plugin
DDP Profiles	Dynamic Device Personalization for Intel® Ethernet 700 Series	https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series
Userspace CNI	Userspace CNI v1.2	https://github.com/intel/userspace-cni-network-plugin
Bond CNI plugin	Bond CNI plugin v1.0	https://github.com/intel/bond-cni
Intel Ethernet Drivers		https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1 https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1 https://sourceforge.net/projects/e1000/files/i40e%20stable/2.7.29 https://sourceforge.net/projects/e1000/files/i40evf%20stable/3.6.15

5.2 Software BOM using Kubernetes v1.15

Table 5. Software BOM using Kubernetes v1.15

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Host OS	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64	https://www.centos.org/
Ansible*	Ansible v2.7.16	https://www.ansible.com/
BMRA Ansible Playbook	v1.3.1	https://github.com/intel/container-experience-kits
Python*	Python 2.7	https://www.python.org/
Kubespray*	Kubespray v2.11	https://github.com/kubernetes-sigs/kubespray
Docker*	Docker 19.03.8	https://www.docker.com/
Container orchestration engine	Kubernetes v1.15.3	https://github.com/kubernetes/kubernetes
CPU Manager for Kubernetes*	CPU Manager for Kubernetes v1.4.0	https://github.com/intel/CPU-Manager-for-Kubernetes

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Telemetry Aware Scheduling	Telemetry Aware Scheduling	https://github.com/intel/telemetry-aware-scheduling
Node Feature Discovery	NFD v0.3.0	https://github.com/kubernetes-sigs/node-feature-discovery
Data Plane Development Kit	DPDK 19.11	https://core.dpdk.org/download/
Open vSwitch with DPDK	OVS-DPDK v2.13.0	http://docs.openvswitch.org/en/latest/intro/install/dpdk/
Vector Packet Processing	VPP 18.07	https://docs.fd.io/vpp/18.07/index.html
Multus CNI	Multus CNI v3.2	https://github.com/intel/multus-cni
SR-IOV CNI	SR-IOV CNI v2.0.0	https://github.com/intel/sriov-cni
SR-IOV network device plugin	SR-IOV network device plugin v3.1	https://github.com/intel/sriov-network-device-plugin
DDP Profiles	Dynamic Device Personalization for Intel® Ethernet 700 Series	https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series
Userspace CNI	Userspace CNI v1.2	https://github.com/intel/userspace-cni-network-plugin
Bond CNI plugin	Bond CNI plugin v1.0	https://github.com/intel/bond-cni
Intel Ethernet Drivers		https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1 https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1 https://sourceforge.net/projects/e1000/files/i40e%20stable/2.7.29 https://sourceforge.net/projects/e1000/files/i40evf%20stable/3.6.15

5.3 Software BOM using Kubernetes v1.16

Table 6. Software BOM using Kubernetes v1.16

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Host OS	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64	https://www.centos.org/
Ansible*	Ansible v2.7.16	https://www.ansible.com/
BMRA Ansible Playbook	v1.4.1	https://github.com/intel/container-experience-kits
Python*	Python 2.7	https://www.python.org/
Kubespray*	Kubespray v2.12.3	https://github.com/kubernetes-sigs/kubespray
Docker*	Docker 19.03.8	https://www.docker.com/
Container orchestration engine	Kubernetes v1.16.7 (includes Topology Manager)	https://github.com/kubernetes/kubernetes
CPU Manager for Kubernetes*	CPU Manager for Kubernetes v1.4.0	https://github.com/intel/CPU-Manager-for-Kubernetes
Telemetry Aware Scheduling	Telemetry Aware Scheduling	https://github.com/intel/telemetry-aware-scheduling
Node Feature Discovery	NFD v0.3.0	https://github.com/kubernetes-sigs/node-feature-discovery
Data Plane Development Kit	DPDK 19.11	https://core.dpdk.org/download/
Open vSwitch with DPDK	OVS-DPDK v2.13.0	http://docs.openvswitch.org/en/latest/intro/install/dpdk/
Vector Packet Processing	VPP 18.07	https://docs.fd.io/vpp/18.07/index.html
Multus CNI	Multus CNI v3.3	https://github.com/intel/multus-cni
SR-IOV CNI	SR-IOV CNI v2.0.0	https://github.com/intel/sriov-cni
SR-IOV network device plugin	SR-IOV network device plugin v3.1	https://github.com/intel/sriov-network-device-plugin
DDP Profiles	Dynamic Device Personalization for Intel® Ethernet 700 Series	https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series
Userspace CNI	Userspace CNI v1.2	https://github.com/intel/userspace-cni-network-plugin
Bond CNI plugin	Bond CNI plugin v1.0	https://github.com/intel/bond-cni

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Intel Ethernet Drivers		https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1 https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1 https://sourceforge.net/projects/e1000/files/i40e%20stable/2.7.29 https://sourceforge.net/projects/e1000/files/i40evf%20stable/3.6.15

5.4 Software BOM Comparison across Kubernetes Versions

The following table lists the appropriate software component versions when used with specific Kubernetes versions.

Table 7. Software BOM Comparison across Kubernetes Versions

SOFTWARE FUNCTION	SOFTWARE COMPONENT		
	Kubernetes v1.14	Kubernetes v1.15	Kubernetes v1.16
Host OS	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64	CentOS* 7.7 build 1908 Kernel version: 3.10.0-1062.9.1.el7.x86_64
Ansible*	v2.7.16	v2.7.16	v2.7.16
BMRA Ansible Playbook	v1.2.1	v1.3.1	v1.4.1
Python*	2.7	2.7	2.7
Kubespray*	v2.10	v2.11	v2.12
Docker*	v19.03.8	v19.03.8	v19.03.8
Container orchestration engine - Kubernetes	v1.14 .6	v1.15.6	v1.16.3 (includes Topology Manager)
CPU Manager for Kubernetes*	v1.4.0	v1.4.0	v1.4.0
Telemetry Aware Scheduling	0da14b0	0da14b0	0da14b0
Node Feature Discovery	v0.3.0	v0.3.0	v0.3.0
Data Plane Development Kit	v19.11	v19.11	v19.11
Open vSwitch with DPDK	v2.13.0	v2.13.0	v2.13.0
Vector Packet Processing	v18.07	v18.07	v18.07
Multus CNI	v3.2	v3.2	v3.3
SR-IOV CNI	v2.0.0	v2.0.0	v2.0.0
SR-IOV network device Plugin	v3.1	v3.1	v3.1
DDP Profiles	Dynamic Device Personalization for Intel® Ethernet 700 Series	Dynamic Device Personalization for Intel® Ethernet 700 Series	Dynamic Device Personalization for Intel® Ethernet 700 Series
Userspace CNI	v1.2	v1.2	v1.2
Bond CNI plugin	v1.0	v1.0	v1.0

5.5 Platform BIOS Settings

Refer to Section 6.1, BIOS Prerequisites for Master and Worker Nodes for the platform BIOS settings used for the reference architecture shown in Figure 1.

6 System Prerequisites

This section describes the minimal system prerequisites needed for the Ansible Host and Kubernetes master nodes and worker nodes. It also provides a list of steps required to prepare hosts for successful deployment. These instructions include:

- Required BIOS/UEFI configuration including virtualization and hyper-threading settings.
- Network topology requirements - list of necessary network connections between the nodes.
- Installation of software dependencies needed to execute Ansible playbooks.
- Generation and distribution of SSH keys that will be used for authentication between Ansible host and Kubernetes cluster target servers.

After satisfying these prerequisites, Ansible playbooks can be downloaded directly from the dedicated GitHub* page (https://github.com/intel/container-experience-kits/releases) or cloned using the Git command-line tool and executed after applying desired the configuration as described in Section 7 in this document.

6.1 BIOS Prerequisites for Master and Worker Nodes

Enter the UEFI or BIOS menu and update the configuration shown in the following table. The deployment in this document uses the BIOS Settings described here: Intel® Speed Select Technology – Base Frequency - Enhancing Performance Application Note

Note: The method for accessing the UEFI or BIOS menu is vendor-specific, for example: https://www.dell.com/support/article/us/en/04/sln167315/how-to-boot-into-the-bios-or-the-lifecycle-controller-on-your-poweredge-server?lang=en

Table 8. 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*
Processor Configuration		Intel® VT-x (Virtualization Technology)	Enable	Enable	*Recommended*
		Intel® HT Technology	Disable	Enable	*Recommended*
		Intel® VT-d (IOMMU, SR-IOV Global Enable)	Enable	Enable	*Required*
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.

6.2 Network Interface Requirements for Master and Worker Nodes

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 worker node 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

6.3 Software Prerequisites for Ansible Host, Master Nodes, and Worker Nodes

1. 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
```

2. Enable passwordless login between all nodes in the cluster.
   a. Create Authentication SSH-Keygen Keys on Ansible Host:

```
# ssh-keygen
```

   b. Upload Generated Public Keys to all the nodes from Ansible Host:

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

7 Deploy Intel Bare Metal Reference Architecture using Ansible Playbook

7.1 Get Ansible Playbook and Modify Variables

1. Get Ansible playbook:

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

   Note: Replace `<version>` with a git tag, release branch name or commit hash to check out the desired version of the scripts.

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]
node4
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 9 as a reference.

```
# vim group_vars/all.yml

## BMRA master playbook variables ##

# Node Feature Discovery
nfd_enabled: true
nfd_build_image_locally: false
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: false
# SR-IOV network device plugin configuration.
# For more information on supported configuration refer to: https://github.com/intel/sriov-
network-device-plugin#configurations
sriovdp_config_data: |
    {
      "resourceList": [{
              "resourceName": "intel_sriov_netdevice",
              "selectors": {
                  "vendors": ["8086"],
                  "devices": ["154c", "10ed"],
                  "drivers": ["iavf", "i40evf", "ixgbevf"]
              }
          },
          {
              "resourceName": "intel_sriov_dpdk",
              "selectors": {
                  "vendors": ["8086"],
                  "devices": ["154c", "10ed"],
                  "drivers": ["vfio-pci"]
              }
          }
      ]
    }

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

# Inetl Telemetry Aware Scheduling
tas_enabled: true
tas_namespace: default
# create default TAS policy: [true, false]
tas_create_policy: true

# Create reference net-attach-def objects
example_net_attach_defs:
```

```
  userspace_ovs_dpdk: false
  userspace_vpp: false
  sriov_net_dp: false

## Proxy configuration ##
#http_proxy: "http://proxy.example.com:1080"
#https_proxy: "http://proxy.example.com:1080"
#additional_no_proxy: ".example.com"

#Topology Manager flags
kubelet_node_custom_flags:
  - "--feature-gates=TopologyManager=true"
  - "--topology-manager-policy=best-effort"

# Kubernetes cluster name, also will be used as DNS domain
cluster_name: cluster.local

## Kubespray variables ##

# default network plugins and kube-proxy configuration
kube_network_plugin_multus: true
multus_version: v3.3
kube_network_plugin: flannel
kube_pods_subnet: 10.244.0.0/16
kube_proxy_mode: iptables
kube_service_addresses: 10.233.0.0/18

# please leave it set to "true", otherwise Intel BMRA features deployed as Helm charts won't be
installed
helm_enabled: true

# Docker registry running on the cluster allows us to store images not avaialble on Docker Hub,
e.g. CMK
registry_enabled: true
registry_storage_class: ""
registry_local_address: "localhost:5000"
```

Table 9. 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
nfd_sleep_interval	30s	NFD sleep interval
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
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

VARIABLE	CHOICES/DEFAULTS	DESCRIPTION
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
sriovdp_config_data: {resourceList:[]}	Config parameters in JSON format	Configure as described here: https://github.com/intel/sriov-network-device-plugin/tree/30e33f1ce2fc7b45721b6de8c8207e65dbf2d508#config-parameters
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
tas_enabled	Boolean, default "true"	Install TAS on the target cluster
tas_namespace	String, default "default"	Kubernetes namespace to be used for TAS deployment
tas_create_policy	Boolean, default "true"	Create default TAS policy
example_net_attach_defs: userspace_ovs_dpdk userspace_vpp sriov_net_dp	Boolean, default "false" Boolean, default "false" Boolean, default "false"	Create reference net-attach-def objects
http_proxy https_proxy no_proxy	<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
kubelet_custom_flags	List of additional Kubelet flags	List of additional flags for Kubelet on each node, by default enables Topology Manager with best-effort policy enabled, available policies: none, best-effort, restricted, single-numa-node
cluster_name	String, default "cluster.local"	Kubespray variable
kube_network_plugin_multus	Boolean, default "true"	Kubespray variable, don't change
multus_version	Git tag, default "v3.3"	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
kube_service_addresses	CIDR string, default "10.233.0.0/18"	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 10 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 example, node1.yml and node2.yml.

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

```
---
# Kubernetes node configuration

# Enable SR-IOV networking related setup
sriov_enabled: false

# sriov_nics: SR-IOV PF specific configuration list
sriov_nics:
  - name: enp24s0f0 # PF interface names
    sriov_numvfs: 2 # number of VFs to create for this PF(enp24s0f0)
```

```
    vf_driver: vfio-pci # VF driver to be attached for all VFs under this PF(enp24s0f0),
# "i40evf", "iavf", "vfio-pci", "igb_uio"
    ddp_profile: "gtp.pkgo" # DDP package name to be loaded into the NIC
  - name: enp24s0f1
    sriov_numvfs: 4
    vf_driver: iavf

sriov_cni_enabled: true

# install DPDK
install_dpdk: true # DPDK installation is required for sriov_enabled:true; default to false

userspace_cni_enabled: true

# Intel Bond CNI plugin
bond_cni_enabled: true

vpp_enabled: true

ovs_dpdk_enabled: true
# CPU mask for OVS-DPDK PMD threads
ovs_dpdk_lcore_mask: 0x1
# Huge memory pages allocated by OVS-DPDK per NUMA node in megabytes
# example 1: "256,512" will allocate 256MB from node 0 abd 512MB from node 1
# example 2: "1024" will allocate 1GB fron node 0 on a single socket board, e.g. in a VM
ovs_dpdk_socket_mem: "256,0"

# Set to 'true' to update i40e and i40evf kernel modules
force_nic_drivers_update: true

# install Intel x700 & x800 series NICs DDP packages
install_ddp_packages: true

# Enables hugepages support
hugepages_enabled: true

# Hugepage sizes available: 2M, 1G
default_hugepage_size: 1G

# Sets how many hugepages of each size should be created
hugepages_1G: 4
hugepages_2M: 0

# CPU isolation from Linux scheduler
isolcpus_enabled: true
isolcpus: "4-7"

# Intel CommsPowerManagement
sst_bf_configuration_enabled: true
# Option sst_bf_mode requires sst_bf_configuration_enabled to be set ‘true’
# There are three configuration modes
# [s] Set SST-BF config (set min/max to 2700/2700 and 2100/2100)
# [m] Set P1 on all cores (set min/max to 2300/2300)
# [r] Revert cores to min/Turbo (set min/max to 800/3900)
sst_bf_mode: s
```

Table 10. Host Variables

VARIABLE	CHOICES/DEFAULTS	DESCRIPTION
sriov_enabled	Boolean, default “true”	Enables SR-IOV VFs for interfaces specified in “sriov_nics” array
sriov_cni_enabled	Boolean, default “true”	Whether to install SR-IOV CNI plugin on target node

VARIABLE	CHOICES/DEFAULTS	DESCRIPTION
sriov_nics: - name sriov_numvfs vf_driver ddp_profile	Array of dicts, please see description	Configure as described here: https://github.com/intel/sriov-network-device-plugin#config-parameters https://github.com/intel/container-experience-kits/blob/master/docs/sriov.md only if "sriov_net_dp_enabled" set to "true"
install_dpdk	Boolean, default "true"	Install DPDK; required, when sriov_enabled set to true
userspace_cni_enabled	Boolean, default "true"	Enables installation of Userspace CNI plugin on target machine
bond_cni_enabled	Boolean, default "true"	Enables installation of Bond 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
install_ddp_packages	Boolean, default "true"	Install Intel x700 & x800 series NICs DDP packages
hugepages_enabled	Boolean, default "true"	Enable hugepage support
default_hugepage_size	"2M" or "1G"	Default hugepage size = 1G. 1G is required for OVS-DPDK and Userspace CNI plugin data plane
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, e.g. "1,2,3-6,8-15,24"	List of CPUs to be isolated
sst_bf_configuration_enabled	Boolean, default "true"	Configure Intel® SST-BF
sst_bf_mode	String, default "s"	Select Intel® SST-BF configuration mode

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.

7.2 Execute the Ansible Playbook

Deploy the Intel Bare Metal Reference Architecture with the command:

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

8 Post-Deployment Verification

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

8.1 Check the Kubernetes Cluster

1. Check the post-deployment node status of the master nodes and worker nodes:

```
# 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 nodes and worker nodes. All pods 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   kube-sriov-device-plugin-amd64-8x46m          1/1     Running             1
4d12h
kube-system   kube-sriov-device-plugin-amd64-lxpns          1/1     Running             1
4d12h
kube-system   kube-sriov-device-plugin-amd64-c8cth          1/1     Running             1
4d12h
kube-system   tiller-deploy-dc85f7fbd-7fn6r                 1/1     Running             3
4d12h
```

8.2 Check Intel® Speed Select Technology – Base Frequency (Intel® SST-BF) Configuration

The Intel® SST-BF feature enables base frequency configuration, which allows some cores to run at a higher guaranteed base frequency than others, if such option is required. It provides three different configuration modes:

- sst_bf_mode: s - set high priority cores to 2700 minimum and 2700 maximum, and set normal priority cores to 2100 minimum and 2100 maximum.
- sst_bf_mode: m - set P1 on all cores (2300 minimum and 2300 maximum).
- sst_bf_mode: r - revert cores to minimum/Turbo (set all cores to 800 minimum and 3900 maximum).

To verify, that Intel® SST-BF was configured as expected, use the following command:

```
# sst-bf.py -i
```

The output should show the current Intel® SST-BF frequency information.

To learn more about Intel® SST-BF, visit https://github.com/intel/CommsPowerManagement.

Note: The Intel® SST-BF feature is not supported on CentOS* 7 when the Linux kernel from the official package repository is used. Before you execute the scripts to configure SST-BF on the target system, you must include mainline Linux kernel v5.1 or newer in the system image or install it manually.

8.3 Check DDP Profiles

DDP provides dynamic reconfiguration of the packet processing pipeline to meet specific use case needs on demand, adding new packet processing pipeline configuration profiles to a network adapter at run time, without resetting or rebooting the server.

To verify that a correct DDP profile was loaded, use the command shown below.

```
# ddptool -a
Intel(R) Dynamic Device Personalization Tool
DDPTool version 1.0.0.0
Copyright (C) 2019 Intel Corporation.

NIC  DevId D:B:S.F      DevName         TrackId  Version      Name
==== ===== ============ =============== ======== ============ ==============================
001) 1572  0000:02:00.0 enp2s0f0        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
002) 1572  0000:02:00.1 enp2s0f1        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
003) 1572  0000:02:00.2 enp2s0f2        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
004) 1572  0000:02:00.3 enp2s0f3        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
005) 154C  0000:03:02.0 N/A             80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
006) 154C  0000:03:02.1 enp3s2f1        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
007) 154C  0000:03:02.2 enp3s2f2        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
008) 154C  0000:03:02.3 N/A             80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
```

Download ddptool at: https://downloads.sourceforge.net/project/e1000/ddptool%20stable/ddptool-1.0.0.0/ddptool-1.0.0.0.tar.gz

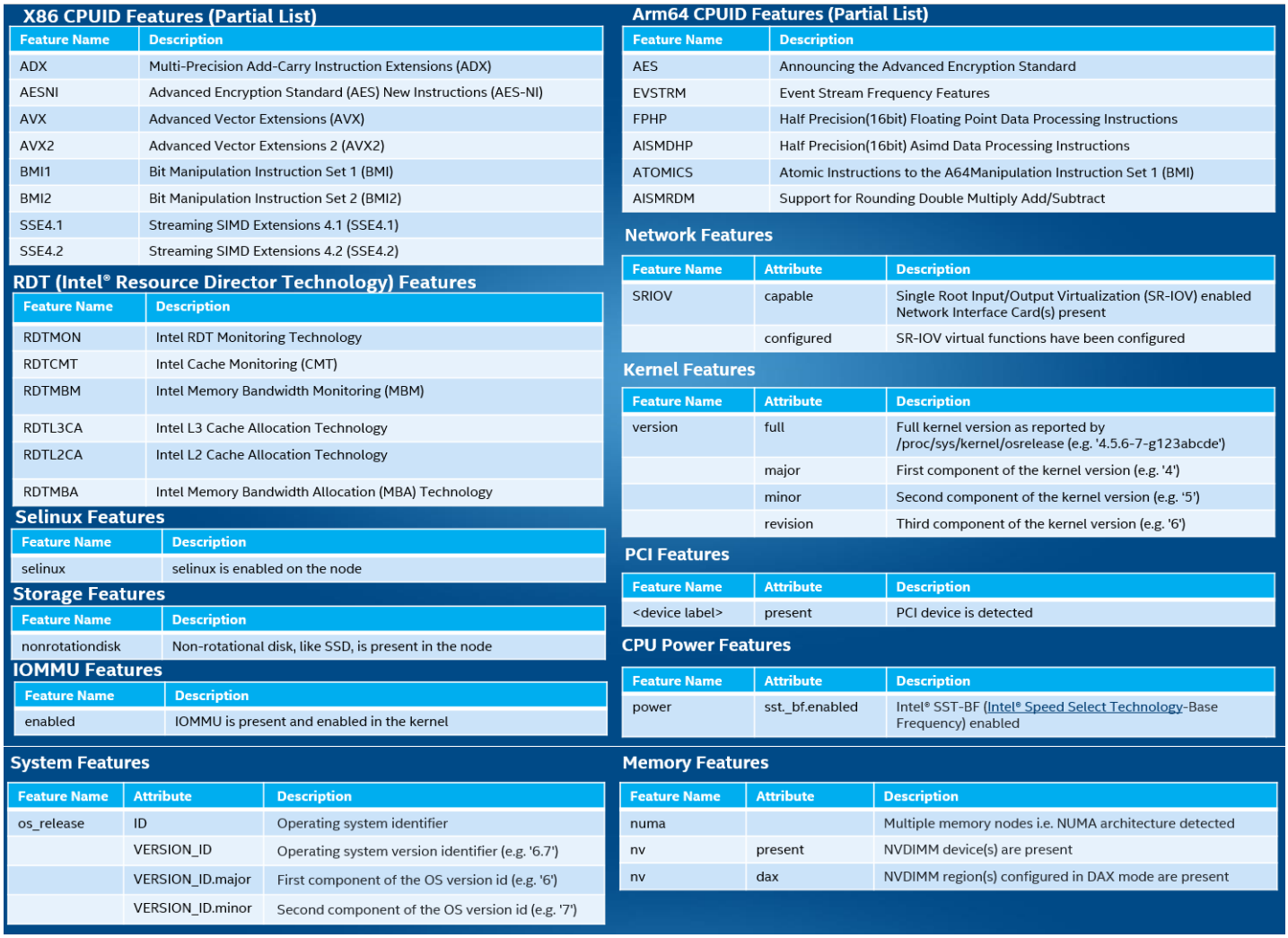
When everything is installed and set up correctly, you will see that the device plugin is able to discover VFs with the DDP profile names given in the resource pool selector.

```
# kubectl get node node1 -o json | jq ".status.allocatable"
{
  "cpu": "8",
  "ephemeral-storage": "169986638772",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "8Gi",
  "intel.com/x700_gtp": "2",
  "intel.com/x700_pppoe": "0",
  "memory": "7880620Ki",
  "pods": "100"
}
```

8.4 Check Node Feature Discovery (NFD)

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. NFD is one of the Intel technologies that supports targeting of 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 below.



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

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

CPU Power Features

Feature Name	Attribute	Description
power	sst_bf.enabled	Intel® SST-BF (Intel® Speed Select Technology-Base Frequency) enabled

System Features

Feature Name	Attribute	Description
os_release	ID	Operating system identifier
	VERSION_ID	Operating system version identifier (e.g. '6.7')
	VERSION_ID.major	First component of the OS version id (e.g. '6')
	VERSION_ID.minor	Second component of the OS version id (e.g. '7')

Memory Features

Feature Name	Attribute	Description
numa		Multiple memory nodes i.e. NUMA architecture detected
nv	present	NVDIMM device(s) are present
nv	dax	NVDIMM region(s) configured in DAX mode are present

Figure 4. 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
```

8.5 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 restricted, 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, Intel® HT is enabled. Therefore both physical and associated logical processors are isolated. Four cores are allocated for the CPU Manager for a 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` 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 is displayed in the format `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": "16Gi",
  "intel.com/intel_sriov_dpdk": "6",
  "intel.com/intel_sriov_netdevice": "6",
  "intel.com/mlnx_sriov_rdma": "0",
  "memory": "179707216Ki",
  "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 Intel technology capabilities to achieve the improved network I/O, deterministic compute performance, and server platform sharing benefits offered by Intel® Xeon® Processor-based platforms.

8.6 Topology Manager

An increasing number of systems leverage a combination of CPUs and hardware accelerators to support latency-critical execution and high-throughput parallel computation. These include workloads in fields such as telecommunications, scientific computing, machine learning, financial services, and data analytics. Such hybrid systems comprise a high-performance environment.

To extract the best performance, required optimizations related to CPU isolation, memory, and device locality must be made. However, in Kubernetes*, these optimizations are handled by a varied set of components. Topology Manager is a Kubelet component for coordinating the set of components that are responsible for these optimizations. It is a native component of Kubernetes v1.16.

Topology Manager supports its allocation policies via a Kubelet flag, `--topology-manager-policy`. There are four supported policies:

- **none:** Kubelet does not perform any topology alignment.
- **best-effort (default in BMRA deployment script):** Using resource availability reported by Hint Providers for each container in a Guaranteed Pod, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager will store this and admit the pod to the node anyway. The Hint Providers can then use this information when making the resource allocation decision.
- **restricted:** Using resource availability reported by Hint Providers for each container in a Guaranteed Pod, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager will reject this pod from the node. This will result in a pod in a Terminated state with a pod admission failure.
  Once the pod is in a Terminated state, the Kubernetes scheduler will not attempt to reschedule the pod. We recommend you use a ReplicaSet or Deployment to trigger a redeploy of the pod. Alternatively, you could implement an external control loop to trigger a redeployment of pods that have the Topology Affinity error.
  If the pod is admitted, the Hint Providers can then use this information when making the resource allocation decision.
- **single-numa-node:** Using resource availability reported by Hint Providers for each container in a Guaranteed Pod, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, Topology Manager will store this and the Hint Providers can then use this information when making the resource allocation decision. If this is not possible, however, then the Topology Manager will reject the pod from the node. This will result in a pod in a Terminated state with a pod admission failure.
  Once the pod is in a Terminated state, the Kubernetes scheduler will not attempt to reschedule the pod. It is recommended to use a Deployment with replicas to trigger a redeploy of the Pod. An external control loop could be also implemented to trigger a redeployment of pods that have the Topology Affinity error.

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

```
# journalctl | grep topologymanager
Dec 04 10:39:27 silpixa00390843 kubelet[9247]: I1204 10:39:27.305994    9247
topology_manager.go:92] [topologymanager] Creating topology manager with best-effort policy
Dec 04 10:39:27 silpixa00390843 kubelet[9247]: I1204 10:39:27.306005    9247
container_manager_linux.go:300] [topologymanager] Initilizing Topology Manager with best-effort
policy
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.050934    9247
topology_manager.go:308] [topologymanager] Topology Admit Handler
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.050942    9247
topology_manager.go:317] [topologymanager] Pod QoS Level:
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.050950    9247
topology_manager.go:332] [topologymanager] Topology Manager only affinitises Guaranteed pods.
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.084236    9247
topology_manager.go:308] [topologymanager] Topology Admit Handler
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.084251    9247
topology_manager.go:317] [topologymanager] Pod QoS Level: BestEffort
Dec 04 10:39:48 silpixa00390843 kubelet[9247]: I1204 10:39:48.084263    9247
topology_manager.go:332] [topologymanager] Topology Manager only affinitises Guaranteed pods.
```

8.6.1 Change Topology Manager Policy: Redeploy Kubernetes Playbook

This section describes one of two ways to change Topology Manager Policy configuration after cluster deployment, by redeploying the Kubernetes playbook.

1. Update the group_vars/all.yml file:

```
...
#Topology Manager flags
kubelet_node_custom_flags:
 - "--feature-gates=TopologyManager=true"
 - "--topology-manager-policy=single-numa-node"
...
```

2. Execute the ansible-playbook command to apply the new configuration cluster-wide:

```
# ansible-playbook -i inventory.ini playbooks/k8s/k8s.yml
```

8.6.2 Change Topology Manager Policy: Manually Update Kubelet Flags

This section describes a method of changing Topology Manager Policy configuration after cluster deployment, by manually updating Kubelet flags on a specific node.

1. Login to the node via SSH, for example:

```
# ssh root@node1
```

2. Edit the KUBELET_ARGS environmental variable in the /etc/kubernetes/kubelet.env file:

```
...
KUBELET_ARGS="--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf \
... (other flags skipped for readability) ...
    --feature-gates=TopologyManager=true  --topology-manager-policy=single-numa-node"
```

```
...
```

3. Restart the kubelet service:

```
# systemctl restart kubelet
```

8.7 Intel Device Plugins for Kubernetes

Like other vendors, Intel provides many hardware devices that help 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.[2] 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

8.7.1 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 the Intel® Ethernet 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 i40evf, iavf 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/intel_sriov_netdevice`

```
# kubectl get node node5 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "8",
  "cpu": "95900m",
  "ephemeral-storage": "48294789041",
  "hugepages-1Gi": "16Gi",
  "intel.com/intel_sriov_dpdk": "6",
  "intel.com/intel_sriov_netdevice": "6",
  "intel.com/mlnx_sriov_rdma": "0",
  "memory": "179707216Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

8.7.2 Check 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 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/products/docs/network-io/ethernet/10-25-40-gigabit-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": "16Gi",
  "intel.com/intel_sriov_dpdk": "6",
  "intel.com/intel_sriov_netdevice": "6",
```

[2] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

```
  "intel.com/mlnx_sriov_rdma": "0",
  "memory": "179707216Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

8.8 Telemetry Aware Scheduling (TAS)

TAS makes telemetry data available to scheduling and descheduling actions in Kubernetes. Through a user defined telemetry scheduling policy, TAS enables rule-based decisions on pod placement powered by up to date platform metrics. Policies can be applied on a workload by workload basis - allowing the right indicators to be used to place the right pod.

Telemetry Aware Scheduling is made up of two components deployed in a single pod on a Kubernetes cluster:

- **Telemetry Aware Scheduler Extender** is contacted by the generic Kubernetes scheduler every time it needs to make a scheduling decision on a pod calling for telemetry scheduling. The extender checks if there is a telemetry policy associated with the workload. If so, it inspects the strategies associated with the policy and returns opinions on pod placement to the generic scheduler. The scheduler extender has two strategies it acts on - `scheduleonmetric` and `dontschedule`. This is implemented and configured as a Kubernetes Scheduler Extender.
- **Telemetry Policy Controller** consumes TAS Policies - a Custom Resource. The controller parses this policy for deschedule, scheduleonmetric and dontschedule strategies and places them in a cache to make them locally available to all TAS components. It consumes new Telemetry Policies as they are created, removes them when deleted, and updates them as they are changed. The policy controller also monitors the current state of policies to see if they are violated

TAS acts on 3 strategy types:

- **scheduleonmetric** has only one rule. It is consumed by the Telemetry Aware Scheduling Extender and prioritizes nodes based on a comparator and an up to date metric value. For example:
  ```
  scheduleonmetric when health_metric is LessThan
  ```
- **dontschedule** has multiple rules, each with a metric name and operator and a target. A pod with this policy will never be scheduled on a node breaking any one of these rules. For example:
  ```
  dontschedule if health_metric Equals 1
  ```
- **deschedule** is consumed by the Telemetry Policy Controller and can have multiple rules. If a pod is running on a node that violates this policy, it can be descheduled with the Kubernetes descheduler. For example:
  ```
  deschedule if health_metric Equals 2
  ```

8.8.1 Verify Telemetry Aware Scheduling

The verification steps below involve a pre-created telemetry scheduling policy which uses a dummy metric called `health metric` for demonstration purposes. This policy is created during TAS installation.

1. Check if a policy has been created:
   ```
   kubectl get taspolicies
   ```
2. Expected output:
   ```
   tas-intel-telemetry-aware-scheduler-scheduling-policy   3d3h
   ```
3. See details of this policy, including the rules and associated metrics:
   ```
   kubectl describe taspolicies tas-intel-telemetry-aware-scheduler-scheduling-policy
   ```
4. Deploy a workload using the created policy:
   ```
   kubectl apply -f /usr/src/telemetry-aware-scheduling/deploy/health-metric-demo/demo-pod.yaml
   ```
   This will cause a pod to be deployed and scheduled using the telemetry scheduling policy from above.
5. Verify that TAS has been called to schedule the pod by looking at the logs:
   ```
   kubectl logs -lapp=tas -ctasext
   ```
   The logs will be similar to:
   ```
   2019/08/19 15:30:59 NODE_B health_metric = 1
   2019/08/19 15:30:59 NODE_A health_metric = 0
   2019/08/19 15:30:59 NODE_A violating : health_metric Equals 1
   2019/08/19 15:30:59 NODE_C health_metric = 0
   2019/08/19 15:30:59 Filtered nodes available for demo-policy : NODE_A NODE_C
   ```

This shows that TAS is up and running and impacting scheduling of pods which call out to it. To explore more complex scheduling and descheduling with TAS, and to learn how to create custom telemetry scheduling policies visit:
https://github.com/intel/telemetry-aware-scheduling

8.9 Verify Networking Features (after Installation)

This section describes how to verify certain CNI plugins.

8.9.1 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 v3.3 has recently been integrated with KubeVirt, officially recognized as a CNCF project, and officially released with Kubespray v2.12. More information and source code can be found at: https://github.com/intel/multus-cni

8.9.2 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 v2.0.0 can be found at: https://github.com/intel/sriov-cni

Verify the networks using the following command:

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

8.9.3 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 data plane 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
```

8.9.4 Bond CNI Plugin

Bond CNI provides a method for aggregating multiple network interfaces into a single bonded interface inside a container in a Kubernetes pod. Linux bonding drivers provide several modes for interface bonding such as round robin and active aggregation.

Bond CNI integrates with Multus and the network attachment definition policy declaration. It can be used alongside Multus and SR-IOV CNI to provide failover for network interfaces in a Kubernetes cluster, to increase the total bandwidth available to a single container interface or for other cases in which interface bonding is used.

To verify that Bond CNI is installed on the host, look for its binary in the /opt/cni/bin directory:

```
# ll /opt/cni/bin/bond
-rwxr-xr-x. 1 root root 3836352 Feb 27 13:03 /opt/cni/bin/bond
```

For more information on how to use Bond CNI, refer to: https://github.com/intel/bond-cni

9 Use Cases

This section provides several typical use cases to utilize the features and advanced networking capabilities 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.[3] 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.

9.1 Use NFD to Select Node for Pod Deployment

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 an 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”
...
```

[3] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

Together with other Intel technologies, including device plugins, NFD facilitates workload optimization through resource-aware scheduling.[4] In particular, NFD can benefit workloads that utilize modern vector data processing instructions, require SR-IOV networking, and have specific kernel requirements.

9.2 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 node (inter-node), because the interface is attached to the NIC via SR-IOV, either directly or through the CNI plugin.

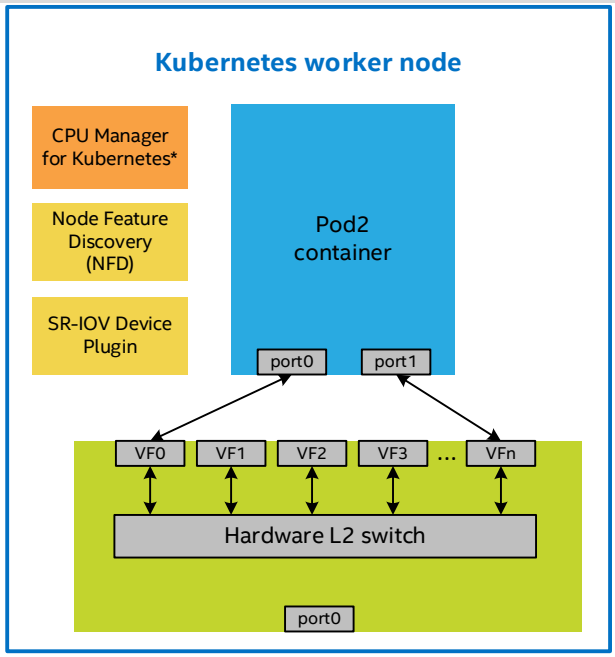
9.2.1 Pod using SR-IOV DPDK

1. Create the pod spec file `sriov-dpdk-res.yaml`.
   In the example below, two VFs are requested from the `intel_sriov_dpdk` 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/intel_sriov_netdevice: '2'
      limits:
        intel.com/intel_sriov_netdevice: '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
```



[4] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

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_INTEL_SRIOV_DPDK=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.

9.2.2 Pod using SR-IOV CNI Plugin

1. Create a sample pod specification `sriov-test.yaml` file.
   In this example, two different network 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.

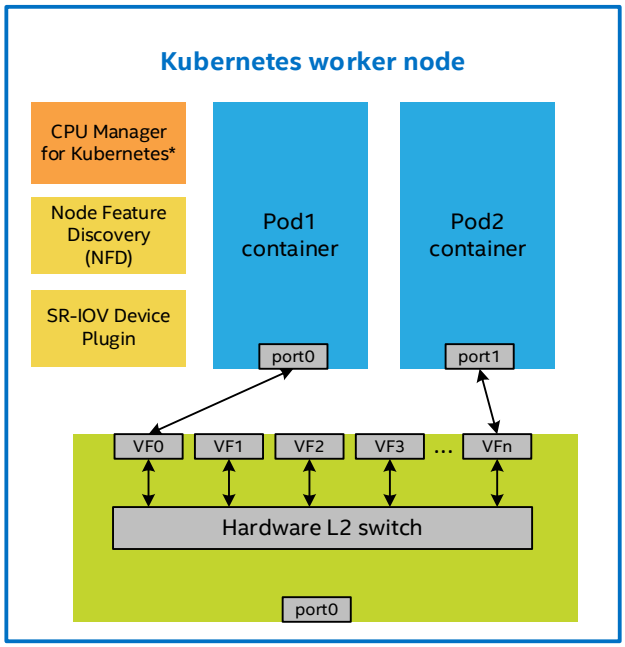
```
# cat sriov-test.yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: sriov-pod-
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "flannel-networkobj" },
            { "name": "sriov-net" }
    ]'
spec:  # specification of the pod's contents
  tolerations:
  - operator: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

9.3 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.[5]

9.3.1 Pod using Userspace CNI with OVS-DPDK

The following example creates a testpmd pod with two interfaces using the `userspace-ovs` 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 hugepage memory resource.

Note: Starting in BMRA Release 1.8, 1G hugepages are available by default. You can change this setting using kernel boot parameters before the system boots up. Refer to Section 7.1 for details.

Here is the testpmd pod specification example file.

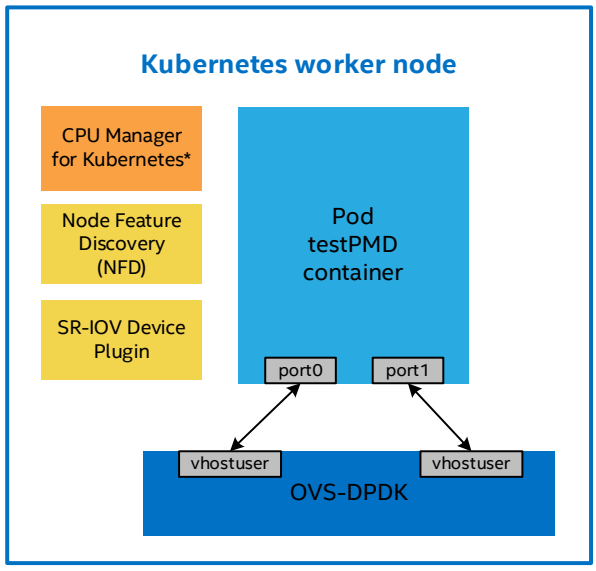
```
# cat testpmd.yaml
apiVersion: v1
kind: Pod
metadata:
  generateName: testpmd-
  annotations:
    k8s.v1.cni.cncf.io/networks: userspace-ovs, userspace-ovs
spec:
  containers:
  - name: testpmd
    image: testpmd:v1.0
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
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

[5] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

```
    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 testpmd 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
  ############################################################################
```

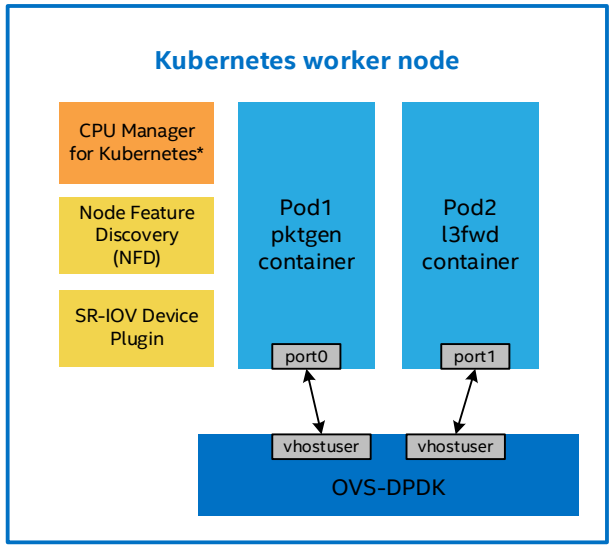
You also can have two or more pods connect the OVS via Userspace CNI plugin. Here is an example using two pods, one is pktgen and one is l3fwd.

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-ovs
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 the `l3fwd.yaml` pod specification , which uses similar resources as `pktgen.yaml`, 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-ovs
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 the master node:

```
# 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 launches. Among the statistics, note 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 the 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>
```

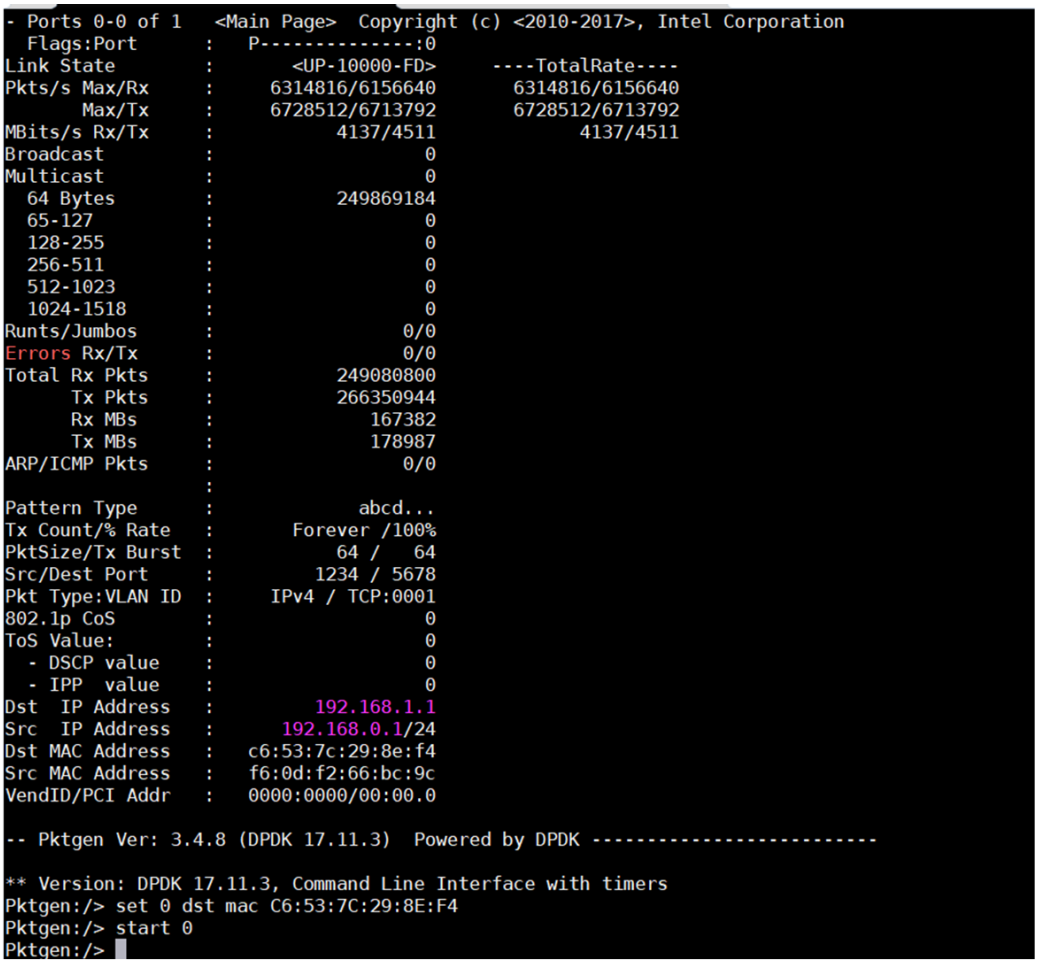
10. Start traffic generation:

```
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
```



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

9.4 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 the previous `testpmd.yaml` file. You can change the core number you want to use for the pod. In the following case, we use 4 exclusive cores.

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

   Note: 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-no-webhook.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 the `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 used along with the other Intel technology capabilities to improve 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 the related suite of changes across the orchestration layers' stack. These Intel technologies enable platform capabilities such as discovery, intelligent configuration and workload-placement decisions resulting in improved and deterministic application performance.

10 Conclusion – Automation Eases Reference Application Deployment

This document contains notes on installation, configuration, and use of 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 playbook enables users to perform automated deployments, which decreases installation time from days to hours. The included example use 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.

Intel and its partners have been working with open source communities to add new techniques and address key barriers to networking adoption in Kubernetes for 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.

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

0420/DN/PTI/PDF 338934-006US