REFERENCE ARCHITECTURE RELEASE V21.09

Intel Corporation

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

Authors

Octavia Carotti

Dave Cremins

Joel A. Gibson

Shivapriya Hiremath

Radoslaw Jablonski

Konrad Janowski

Veronika Karpenko

Patryk Klimowicz

Krystian Mlynek

Dana Nehama

Michael O'Reilly

Michael Pedersen

Dmitrii Puzikov

Syed Faraz Ali Shah

1 Introduction

The **Container Bare Metal Reference Architecture (BMRA)** is a cloud-native, forward-looking Kubernetes-cluster template solution for network implementations. This guide provides instructions on how to set, install, provision, and run a Container Bare Metal Kubernetes cluster setup based on Intel® platforms and open-source software. In addition, the guide provides access to a set of open-source Ansible scripts that enable automatic and easy provisioning of the BMRA using optimized configurations, thus, decreasing the installation time from days to a few hours.

The document provides the following information:

- Part 1 (Section 1-2) provides an overview of the BMRA structure, taxonomy, hardware, and software supported technology options.
- Part 2 (Sections 3 – 7) provides more profound technology and implementation details.
- **If you wish to start building your BMRA right away, you may go directly to Part 3 of this document** (Appendixes A – F) and start automatically provision the BMRA configuration of your choice.
- Part 4 (Appendix G) includes the BMRA 21.09 Release Notes.
- Part 5 (Appendix H) contains an example of Key Management implementation with NGINX.

This guide describes the set of hardware and open-source software components forming the BMRA Release 21.09. The hardware platforms supported are based on the **3rd Generation Intel® Xeon® Scalable processors**, Intel accelerators, and other advanced Intel platform technologies. The provisioning of BMRA Kubernetes clusters is completely automated, using open-source Ansible scripts.

Network deployments vary by location. Each location imposes different hardware and software specifications and configurations due to varying workloads, cost, density, and performance requirements. To address these matters, the BMRA supports the concept of Configuration Profiles, each of which determines how a BMRA flavor can be generated. BMRA 21.09 continues to support the following Reference Architecture Configuration Profiles. Three of the Reference Architecture Configuration Profiles are network location-specific:

- **On-Premises Edge Configuration Profile** – Typical Customer Premises deployment supporting, for example, Content Delivery Network (CDN) and Smart City scenarios.
- **Remote Central Office-Forwarding Configuration Profile** – Near Edge deployments supporting fast packet forwarding workloads such as Cable Modem Termination System (CMTS), Virtual Broadband Network Gateway (vBNG), and User Plane Function (UPF).
- **Regional Data Center Configuration Profile** – Central-office location typical Configuration Profile. Currently tailored exclusively for Media Visual Processing workloads such as CDN Transcoding.

Two additional Reference Architecture Configuration Profiles that are not location-specific enable flexible deployments per need:

- **Basic Configuration Profile** – Minimum BMRA Kubernetes cluster setup.
- **Full Configuration Profile** – Complete BMRA setup supporting all available software features.

A set of Ansible scripts is used to automatically generate and configure BMRA per the Reference Architecture Configuration Profiles. **The guide provides a step-by-step process and describes the functions implemented by each of the Ansible scripts** and goes into great detail to describe the software capabilities and the optimized hardware and software configurations that are implemented by those Ansible scripts.

BMRA Release 21.09 continues the evolution seen in previous BMRA releases and delivers a significant portfolio of technologies that enable a variety of on-premises to core network use cases addressing: security, media, cable (Virtual Cable Modem Termination System (vCMTS)), central office (Access Gateway Function (AGF), Virtual Broadband Network Gateway (vBNG), 5G User Plane Function (UPF)), and Smart City deployments. For the complete list of hardware and software capabilities offered by BMRA, visit Section 2 of this document.

Following are the primary new and updated software features introduced in Release 21.09. For the more complete list, including bug fixes, refer to the BMRA Release Notes.

- Support for microservices architectures with Istio service mesh, Istio operator, and Envoy
- Support for Telegraf platform and application telemetry collection (in addition to collectd)
- Support Intel Telemetry Insight Reports that translate raw platform metrics into up-leveled networking and operational insights (add-on software available under NDA)
- Support CRI-O Kubernetes container runtime interface to enable using Open Container Initiative (OCI)-compatible runtimes (in addition to Docker and containerd)
- Support Calico as the default network plugin
- Support an enhanced set of Intel® Speed Select technologies [in addition to the existing support for Intel® Speed Select Technology – Base Frequency (Intel® SST-BF) and Intel® Speed Select Technology – Core Power (Intel® SST-CP)]:
  - Intel® Speed Select Technology - Performance Profile (Intel® SST-PP) provides a set of performance profiles that ease defining core throughput and determinism
  - Intel® Speed Select Technology – Turbo Frequency (Intel® SST-TF) looks to optimize opportunistic excursions into turbo frequencies
- Support for rendering BMRA Configuration Profile files from template
- Updated Intel® Ethernet 700 and 800 Network Adapter drivers
- Updated Intel® Software Guard Extensions (Intel® SGX) Software Development Kit (SDK)
- Updated Data Plane Development Kit (DPDK) 21.08 and Open vSwitch (OVS) DPDK 2.16, including optimizations for Intel® AVX-512 instructions
- Updated Prometheus, Grafana, and Node Exporter telemetry packages
- Updated Kubernetes to v1.21.x
- Updated Node Feature Discovery (NFD)
- Updated Multus container network interface (CNI)
- Updated OpenSSL toolkit
- Updated Intel® QuickAssist Technology Engine for OpenSSL (Intel® QAT Engine for OpenSSL)
- Updated Intel® Multi-Buffer Crypto for IPSec (intel-ipsec-mb)

Experience Kits, the collaterals that explain in detail the technologies enabled in BMRA release 21.09, including benchmark information, are available in the following locations:

Intel Network Builder:

- https://networkbuilders.intel.com/intel-technologies/network-transformation-exp-kits
- https://networkbuilders.intel.com/intel-technologies/container-experience-kits
- https://networkbuilders.intel.com/intel-technologies/3rd-gen-intel-xeon-scalable-processors-experience-kits

Table of Contents

Figures

Tables

Document Revision History

There are two previous editions of the BMRA document. The editions were released starting April 2019.

- Covered 2nd Generation Intel® Xeon® Scalable processors
- Covered 2nd Generation and 3rd Generation Intel® Xeon® Scalable processors

REVISION	DATE	DESCRIPTION
001	November 2020	Initial release of the BMRA document for 2nd Generation and 3rd Generation Intel® Xeon® Scalable processors.
002	February 2021	Added Appendix for Ansible Host setup. Updated software versions. Updated the release notes to include details about bug fixes for Release 2.1 of the Container Bare Metal Reference Architecture (BMRA).
003	April 2021	Added Appendix for BMRA Regional Data Center Configuration Profile Setup. Added Appendix for Workloads and Application Examples. Introduced new hardware and software technologies and upgrades, including Intel® Server Graphics 1 support. Updated the release notes to include details about bug fixes for BMRA Release 21.03. Revised the document for public release to Intel® Network Builders.
004	April 2021	Added “Check Intel® Server GPU Device and Driver” section.
005	August 2021	Added support for containerized Intel® SGX Key Management Service (KMS), OpenSSL, QAT Engine, and the addition of containerd runtime. Includes support for two operators, SR-IOV and Intel® Device Plugin.
006	October 2021	Updated with information for the 21.09 release, which includes support for Istio service mesh and improved telemetry.
007	October 2021	Added information about Data Plane Development Kit (DPDK) and Open vSwitch (OVS) DPDK to the Introduction.

1.1 About this Document

The BMRA document is composed of five parts, which are described below.

Note: This document goes into great detail to describe the hardware and software ingredients and the configuration options for each BMRA Configuration Profile. Be aware that most of these data points are informational because the setup of each resulting BMRA Flavor is executed automatically by an Ansible playbook.

Part 1 (Sections 1 and 2 in this guide): The guide begins with an overview of the container Bare Metal Reference Architecture (BMRA), including an introduction to the new concept of Reference Architecture Configuration Profiles, use cases supported, a description of software capabilities, and an overview of the Ansible scripts that enable automatic deployment.

Part 2 (Sections 3 – 7 in this guide): This is the main part of the guide. It provides a complete and detailed description of the BMRA components and configuration values, and the processes used for deployment and verification.

- The Ansible playbooks that generate the BMRA Flavors by implementing the BMRA Configuration Profiles are explained. The basic building blocks of Ansible playbooks are described, along with details about how to use them for deployment.
- The next chapter explains all the common software capabilities, including Kubernetes features, packet-processing, telemetry, and others, so that you can evaluate which ones are appropriate for your scenario.
- The following chapters provide a complete description of the BIOS setting options, hardware and software components consumed by the BMRA, and the hardware and software configuration values set by the Ansible playbooks.
- The Post Deployment Verification Guidelines chapter describes a set of processes that you can use to verify the components deployed by the scripts.

Part 3 (Appendixes A – F): Build your BMRA Flavor using customized instructions for each BMRA Configuration Profile.

Those wanting to start right away building a BMRA Flavor can start with Part 3 of the document. The appendixes provide step-by-step instructions on how to generate and deploy a BMRA Flavor according to a specific Configuration Profile. Each appendix contains the hardware and software BOMs and complete details for configuring and executing the appropriate playbook.

Part 4 (Appendix G): Release Notes for BMRA V21.09.

The release notes provide an overview of the new capabilities, bug fixes, and known issues for this BMRA release.

Part 5 (Appendix H): Workloads and Application Examples.

This part provides examples on how to provision and deploy example applications or workloads. For BMRA 21.09, instructions are provided for a Key Management reference application.

1.2 Terminology

Here are some key concepts and terminology used throughout this document:

- A **Reference Architecture** provides a template solution for a specific implementation, typically using industry best practices and the latest technology developments.
- **Container Bare Metal Reference Architecture (BMRA)** – A Reference Architecture that implements a containers bare metal deployment model of a Kubernetes cluster. This guide provides an Intel implementation of a Kubernetes cluster that supports containers on a bare metal platform using open-source software.
- **Reference Architecture Configuration Profile** – A Configuration Profile defines (1) a specific set of hardware ingredients used to build a Kubernetes cluster (the hardware BOM), (2) software modules and capabilities that enable the system design (the software BOM), and (3) specific configuration of those hardware and software elements. In this document we discuss BMRA Configuration Profiles that have been defined and optimized per network location.
- A **Reference Architecture Flavor** is an instance of a Reference Architecture generated by implementing a Configuration Profile specification. In this document we discuss a defined set of BMRA Flavors defined per network location.
- **Reference Architecture Ansible Playbook** – A set of scripts that prepare, configure, and deploy your Kubernetes cluster. This document discusses Reference Architecture Ansible scripts designed for BMRA. The Ansible scripts implement the BMRA Configuration Profiles and generate BMRA Flavors per network location.
- A Kubernetes cluster contains at least one *worker node* that runs containerized applications. *Pods* are the components of the application workload that are hosted on worker nodes. *Control nodes* manage the pods and worker nodes.

Table 1. Abbreviations

ABBREVIATION	DESCRIPTION
ADI	Ad Insertion
AGF	Access Gateway Function

ABBREVIATION	DESCRIPTION
AIA	Accelerator Interfacing Architecture
AMX	Advance Matrix Multiply
AV1	AOMedia Video 1 video coding format
AVC	Advanced Video Coding video compression standard
BIOS	Basic Input/Output System
BKC	Best Known Configuration
BMRA	Bare Metal Reference Architecture
CA	Certificate Authority
CDN	Content Delivery Network
CLOS	Class of Service
CMK	CPU Manager for Kubernetes
CMTS	Cable Modem Termination System
CNCF	Cloud Native Computing Foundation
CNF	Cloud Native Network Function
CNI	Container Network Interface
CO	Central Office
CPI	Cycles Per Instruction
CRD	Custom Resource Definition
CRI	Container Runtime Interface
CSR	Certificate Signing Request
CXL	Compute Express Link
DDP	Dynamic Device Personalization
DHCP	Dynamic Host Configuration Protocol
DLB	Intel® Dynamic Load Balancer (Intel® DLB)
DNS	Domain Name Service
DoS	Denial of Service
DP	Device Plugin
DPDK	Data Plane Development Kit
DRAM	Dynamic Random Access Memory
FP	Floating Point
FPGA	Field-Programmable Gate Array
FW	Firmware
GPU	Graphics Processor Unit
GRUB	GRand Unified Bootloader
HA	High Availability
HCC	High Core Count
HEVC	High Efficiency Video Coding video compression
HSM	Hardware Security Model
HT	Hyper Threading
IA	Intel® architecture
IAX	In-Memory Analytics
IMC	Integrated Memory Controller
Intel® AVX	Intel® Advanced Vector Extensions (Intel® AVX)
Intel® AVX-512	Intel® Advanced Vector Extension 512 (Intel® AVX-512)
Intel® DLB	Intel® Dynamic Load Balancer (Intel® DLB)
Intel® HT Technology	Intel® Hyper-Threading Technology (Intel® HT Technology)

ABBREVIATION	DESCRIPTION
Intel® QAT	Intel® QuickAssist Technology (Intel® QAT)
Intel® RDT	Intel® Resource Director Technology (Intel® RDT)
Intel® Scalable IOV	Intel® Scalable I/O Virtualization (Intel® Scalable IOV)
Intel® SecL – DC	Intel® Security Libraries for Data Center (Intel® SecL – DC)
Intel® SGX	Intel® Software Guard Extensions (Intel® SGX)
Intel® SST-BF	Intel® Speed Select Technology – Base Frequency (Intel® SST-BF)
Intel® SST-CP	Intel® Speed Select Technology – Core Power (Intel® SST-CP)
Intel® SST-PP	Intel® Speed Select Technology – Performance Profile (Intel® SST-PP)
Intel® SST-TF	Intel® Speed Select Technology – Turbo Frequency (Intel® SST-TF)
Intel® VT-d	Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d)
Intel® VT-x	Intel® Virtualization Technology (Intel® VT) for IA-32, Intel® 64 and Intel® Architecture (Intel® VT-x)
IOMMU	Input/Output Memory Management Unit
IPC	Instructions per Cycle
ISA	Instruction Set Architecture
I/O	Input/Output
K8s	Kubernetes
KMRA	Key Management Reference Application (KMRA)
KMS	Key Management Service (KMS)
LCC	Low Core Count
LLC	Last Level Cache
LOM	LAN on Motherboard
MMIO	Memory-Mapped Input/Output
MPEG-2	Moving Picture Experts Group standard for digital television and DVD video
MSR	Model-Specific Register
NF	Network Function
NFD	Node Feature Discovery
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NIC	Network Interface Card
NTB	Non-Transparent Bridge
NTP	Network Time Protocol
NVM	Non-Volatile Memory
NVMe	Non-Volatile Memory
OAM	Operation, Administration, and Management
OCI	Open Container Initiative
OS	Operating System
OVS	Open vSwitch
OVS DPDK	Open vSwitch with DPDK
PBF	Priority Based Frequency
PCCS	Provisioning Certification Caching Service
PCI	Physical Network Interface
PCIe	Peripheral Component Interconnect express
PF0	First physical function of the device
PKCS	Public-Key Cryptography Standard
PMD	Poll Mode Driver
PMU	Power Management Unit

ABBREVIATION	DESCRIPTION
PSP	Pod Security Policy
PXE	Preboot Execution Environment
QAT	Intel® QuickAssist Technology
QoS	Quality of Service
RA	Reference Architecture
RAN	Resource Allocation Network
RAS	Reliability, Availability, and Serviceability
RBAC	Role-Based Access Control
RDT	Intel® Resource Director Technology
S-IOV	Intel® Scalable I/O Virtualization (Intel® Scalable IOV)
SA	Service Assurance
SDN	Software-Defined Networking
SGX	Intel® Software Guard Extensions (Intel® SGX)
SHVS	Standard High-Volume Servers
SIMD	Single Instruction, Multiple Data
SMTC	Smart City
SoC	System on Chip
SOCKS	Socket Secure
SR-IOV	Single Root Input/Output Virtualization
SSD	Solid State Drive
SSH	Secure Shell Protocol
SVM	Shared Virtual Memory
TAS	Telemetry Aware Scheduling
TDP	Thermal Design Power
TLS	Transport Layer Security
TME	Total Memory Encryption
TMUL	Tile Multiply
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface
UPF	User Plane Function
vBNG	Virtual Broadband Network Gateway
vCDN	Virtualized Content Delivery Network
vCMTS	Virtual Cable Modem Termination System
VF	Virtual Function
VLAN	Virtual LAN
VMM	Virtual Machine Manager
VNF	Virtual Network Function
VP9	Video coding format for streaming over the internet
VPP	Vector Packet Processing
VXLAN	Virtual Extensible LAN

1.3 Taxonomy

The following table describes the terminology used in the other tables in this document.

Table 2. Hardware and Software Taxonomy

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

1.4 Reference Documents

Collaterals, including technical guides and solution briefs that explain in detail the technologies enabled in BMRA release 21.09, are available in the following locations: https://networkbuilders.intel.com/intel-technologies/network-transformation-exp-kits and https://networkbuilders.intel.com/intel-technologies/container-experience-kits.

Table 3. 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
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
Enhanced Utilization Telemetry for Polling Workloads with collectd and the Data Plane Development Kit (DPDK) User Guide	https://networkbuilders.intel.com/solutionslibrary/enhanced-utilization-telemetry-for-polling-workloads-with-collectd-and-the-data-plane-development-kit-dpdk-user-guide
Intel Device Plugins for Kubernetes Application Note	https://builders.intel.com/docs/networkbuilders/intel-device-plugins-for-kubernetes-appnote.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
Intel® Ethernet Controller 700 Series GTPv1 - Dynamic Device Personalization Application Note	https://networkbuilders.intel.com/solutionslibrary/intel-ethernet-controller-700-series-gtpv1-dynamic-device-personalization
Intel® Ethernet Controller E810 Dynamic Device Personalization Package (DDP) for Telecommunications Technology Guide	https://www.intel.com/content/www/us/en/products/details/ethernet/800-controllers/e810-controllers/docs.html
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
Node Feature Discovery Application Note	https://builders.intel.com/docs/networkbuilders/node-feature-discovery-application-note.pdf
QCT Validated Kubernetes Platform with Enhanced Platform Awareness Technical Brief	https://networkbuilders.intel.com/solutionslibrary/qct-validated-kubernetes-platform-with-enhanced-platform-awareness
Telemetry Aware Scheduling (TAS) - 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
Topology Management - Implementation in Kubernetes Technology Guide	https://builders.intel.com/docs/networkbuilders/topology-management-implementation-in-kubernetes-technology-guide.pdf
Intel® Speed Select Technology – Base Frequency (Intel® SST-BF) with Kubernetes Application Note	https://networkbuilders.intel.com/solutionslibrary/intel-speed-select-technology-base-frequency-with-kubernetes-application-note
Secure the Network Infrastructure - Secure Cloud Native Network Platforms User Guide	https://networkbuilders.intel.com/solutionslibrary/secure-the-network-infrastructure-secure-cloud-native-network-platforms-user-guide

REFERENCE	SOURCE
Closed Loop Automation - Telemetry Aware Scheduler for Service Healing and Platform Resilience Demo	https://networkbuilders.intel.com/closed-loop-automation-telemetry-aware-scheduler-for-service-healing-and-platform-resilience-demo
Closed Loop Automation - Telemetry Aware Scheduler for Service Healing and Platform Resilience White Paper	https://builders.intel.com/docs/networkbuilders/closed-loop-platform-automation-service-healing-and-platform-resilience.pdf
Intel Telemetry Insights Reports (For access, contact your Intel Platform Application Engineer)	https://www.intel.com/content/www/us/en/secure/design/internal/content-details.html?DocID=645751
Intel® RDT	https://wiki.opnfv.org/display/fastpath/Intel_RDT
Intel® Server GPU Features	https://www.intel.com/content/www/us/en/products/discrete-gpus/server-graphics-card.html
Intel® Server GPU for High-Density Cloud Gaming and Media Streaming Performance Summary	https://www.intel.com/content/www/us/en/benchmarks/server/graphics/intelservergpu.html
Intel® Server GPU Data Center Graphics for High-Density Cloud Gaming and Media Streaming	https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/server-graphics-card-product-brief.html
Intel® Server GPUs for Cloud Gaming and Media Delivery	https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/server-graphics-solution-brief.html
Intel® Server GPU Specifications	https://ark.intel.com/content/www/us/en/ark/products/210576/intel-server-gpu.html
Intel® AVX-512 - Packet Processing with Intel® AVX-512 Instruction Set Solution Brief	https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-packet-processing-with-intel-avx-512-instruction-set-solution-brief
Intel® AVX-512 - Instruction Set for Packet Processing Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-instruction-set-for-packet-processing-technology-guide
Intel® AVX-512 - Writing Packet Processing Software with Intel® AVX-512 Instruction Set Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-writing-packet-processing-software-with-intel-avx-512-instruction-set-technology-guide
Crypto - Ubiquitous Availability of Crypto Technologies Solution Brief	https://networkbuilders.intel.com/solutionslibrary/ubiquitous-availability-of-crypto-technologies-solution-brief
Intel® Architecture Instruction Set Extensions and Future Features Programming Reference	https://wiki.ith.intel.com/download/attachments/1508654814/architecture-instruction-set-extensions-programming-reference.pdf?version=1&modificationDate=1581680607737&api=v2
Intel® Software Guard Extensions (Intel® SGX) – Key Management on the 3rd Generation Intel® Xeon® Scalable Processor Technology Guide	https://networkbuilders.intel.com/solutionslibrary/intel-software-guard-extensions-intel-sgx-key-management-on-the-3rd-generation-intel-xeon-scalable-processor-technology-guide
Intel® Software Guard Extensions (Intel® SGX) – NGINX Private Key on 3rd Generation Intel® Xeon® Scalable Processor User Guide	https://networkbuilders.intel.com/solutionslibrary/intel-software-guard-extensions-intel-sgx-nginx-private-key-on-3rd-generation-intel-xeon-scalable-processor-user-guide

2 Reference Architecture Overview

This chapter provides an overview of the Reference Architecture itself, including high-level descriptions of the Configuration Profiles, use cases, hardware components, software capabilities, and Ansible playbooks.

Review Section 1.1, which introduces key concepts and terminology used in this guide.

2.1 Architecture Delivered

The Container Bare Metal Reference Architecture (BMRA) is an open Kubernetes cluster architecture designed for the convergence of key applications and services, control plane, and high-performance packet processing functions. It enables adoption of Intel platforms (starting with Early Access hardware platforms) and open-source platform software capabilities.

This complete, flexible, and scalable solution template allows deployment of Kubernetes clusters that can be based on multiple worker nodes managed by one or more Kubernetes control nodes. All servers are connected with one or two switches that provide connectivity within the cluster and to the cloud. The Ansible playbook allows auto-provisioning and auto-configuration of the BMRA and can be installed on any connected host server. In addition, this Reference Architecture uses testpmd and pktgen pods and sample CNF workloads (vCMTS and vBNG) for integration validation.

Figure 1 below provides an example of a typical Kubernetes cluster composed of three control nodes and two worker nodes.

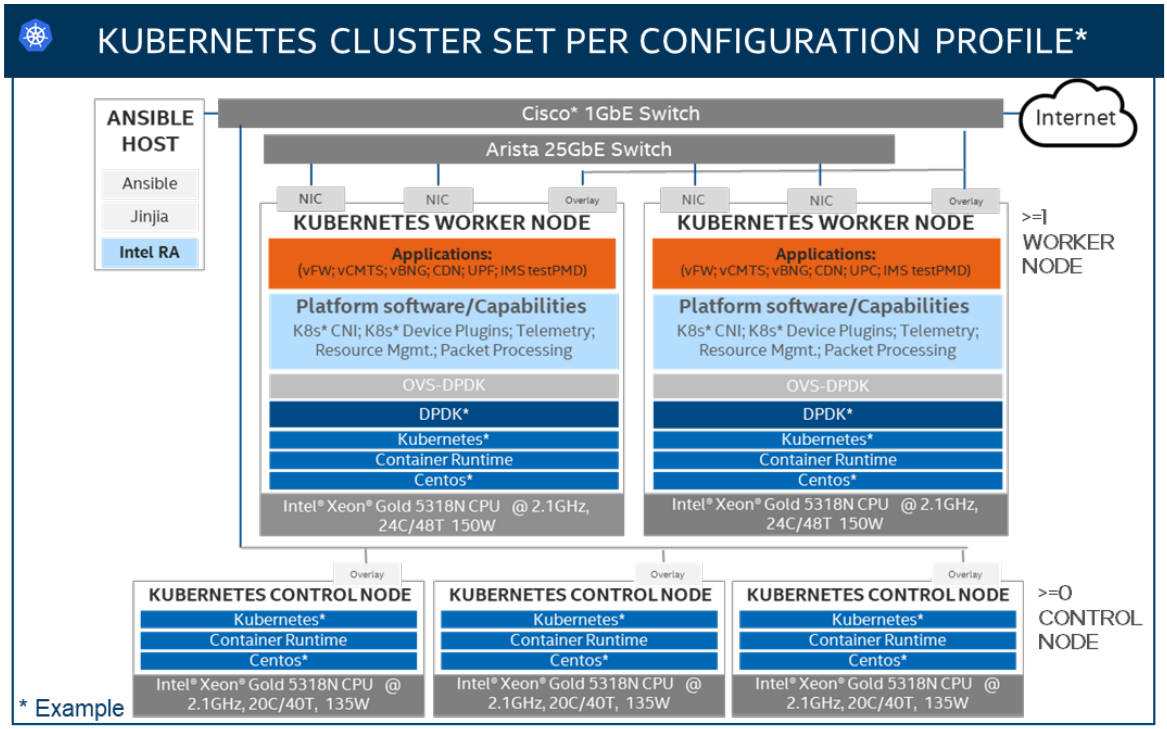

Figure 1. Example of Container Bare Metal Reference Architecture Kubernetes Cluster Setup[1]

The main elements forming the reference architecture are:

- **Hardware Components**: Multiple platform hardware options are available, including a variety of 2nd Generation Intel® Xeon® Scalable processor SKUs, 3rd Generation Intel® Xeon® Scalable processor SKUs, and Intel® Ethernet Network Adapters. For details see Section 5, Reference Architecture Hardware Components and BIOS.
  BIOS options are provided, and you are expected to select and deploy the most optimal BIOS values before the cluster provisioning. For details, see Section 5.5, Platform BIOS.
- **Software Capabilities**: This container environment uses Docker containers runtime as well as containerd and cri-o. The software capabilities are based on open-source software enabled in communities such as DPDK, FD.io. OVS, OVS-DPDK Kubernetes, and through Intel GitHub. Three Linux-based operating system options are available: CentOS, RHEL, and Ubuntu. For details, see Section 6, Reference Architecture Software Components.
- **Configuration**: Specific hardware and software configurations are provided based on Intel assessment and verification. The hardware configuration addresses two modes of operation: “base” and “plus” (for high performance).
- **Installation Playbook**: Ansible playbooks implement the best practice configuration setup per each BMRA Flavor. This guide provides great details about the hardware and software configuration options available; however, the Ansible playbooks are ready to use, automate, and shorten the setup of your BMRA Flavor of choice.

[1] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products. See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

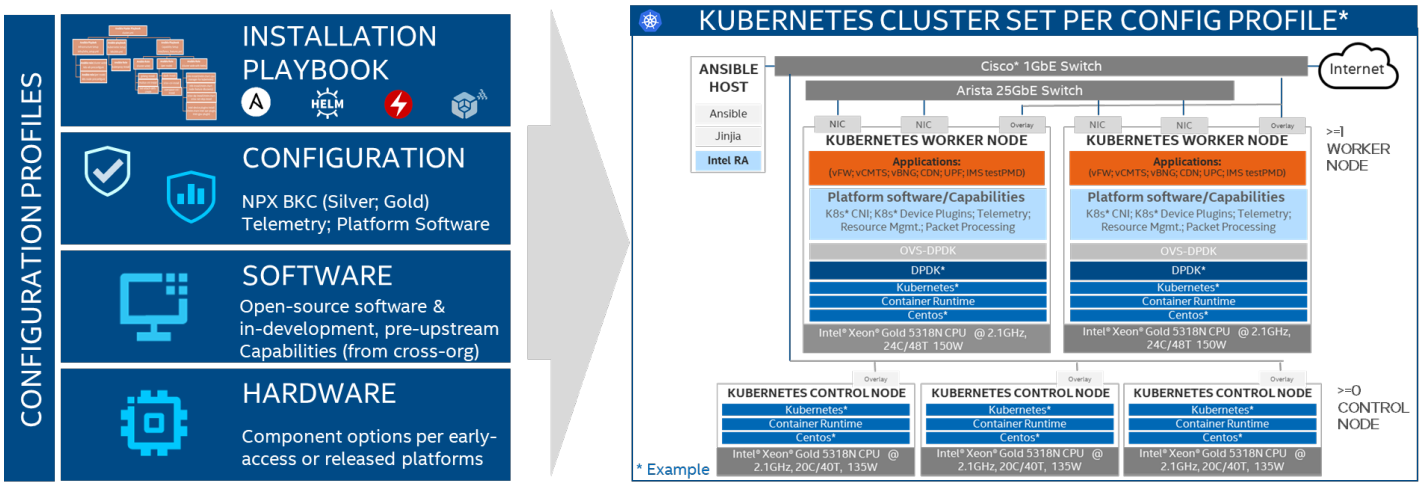

Figure 2. Key Components forming the Container Bare Metal Reference Architecture Kubernetes Cluster

2.2 Reference Architecture Configuration Profiles

Deployment requirements across the network are different depending on the location in the network and on the typical applications supported at each location. Each location can be characterized by the sites' density, distance from the user, and performance (latency and throughput). We identified the following main "network locations" as shown in Figure 3: **On-Premises Edge** (also called Customer Premise), **Central Office** (also called Near Edge), **Regional Data Center** (also called Regional POP), and **Telco Data Center Core Network**.

The BMRA introduces the concept of **Reference Architecture Configuration Profiles**. Multiple Configuration Profiles are assigned to the Reference Architecture to address the specific configuration, performance, and functionality required per workload type and network location. The following Configuration Profiles are supported and are shown in Figure 4.

- **On-Premises Edge Configuration Profile**: This Configuration Profile recommends a K8s cluster hardware configuration, software capabilities, and specific hardware and software configurations that typically support enterprise edge workloads used in SMTC deployments, CDN, and Ad-insertion. Refer to Appendix D, BMRA On-Premises Edge Configuration Profile Setup for details.
- **Remote CO-Forwarding Configuration Profile**: This Configuration Profile addresses a K8s cluster hardware, software capabilities, and configurations that enable high performance for packet forwarding packets. In this category, you can find workloads such as UPF, vBNG, vCMTS, vCDN, FW, and more. The corresponding Configuration Profile is described in Appendix E, BMRA Remote CO-Forwarding Configuration Profile Setup.
- **Regional Data Center Configuration Profile –** This Configuration Profile is tailored exclusively and defined for Media Visual Processing workloads such as CDN Transcoding. The corresponding Configuration Profile is described in Appendix F BMRA Regional Data Center Configuration Profile Setup.

In addition, we have identified the following Configuration Profiles that are not specific to network location or workload:

- **Basic Configuration Profile**: Minimum hardware and software capabilities required for supporting a BMRA Kubernetes cluster setup, described in Appendix B, BMRA Basic Configuration Profile Setup.
- **Full Configuration Profile**: Complete hardware and software capabilities set offered in BMRA, described in Appendix C, BMRA Full Configuration Profile Setup.

Key elements of each Reference Architecture Configuration Profile include:

1. Example Use Cases: Scenarios that support the specific workloads enabled by the Configuration Profile.
2. Installation Playbook: An Ansible script enabling the deployment of all hardware and software components with optimized configurations.
3. Hardware Components (Section 5, Reference Architecture Hardware Components and BIOS): a set of optimized hardware elements for the location. For example, to support UPF workload at the Remote CO, the BMRA is populated with the maximum available network interface cards (NICs).
4. Software Capabilities (Section 6, Reference Architecture Software Components): The software components and the configuration per feature.

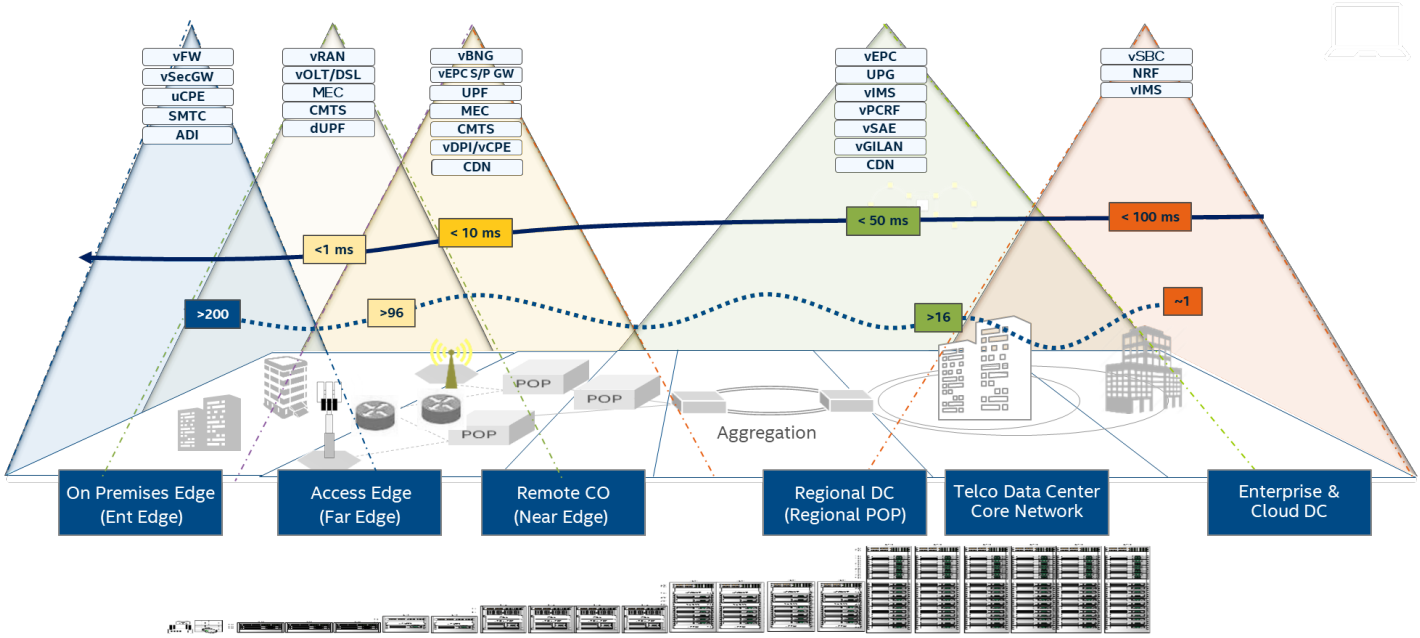

Figure 3. Identified Key Network Locations

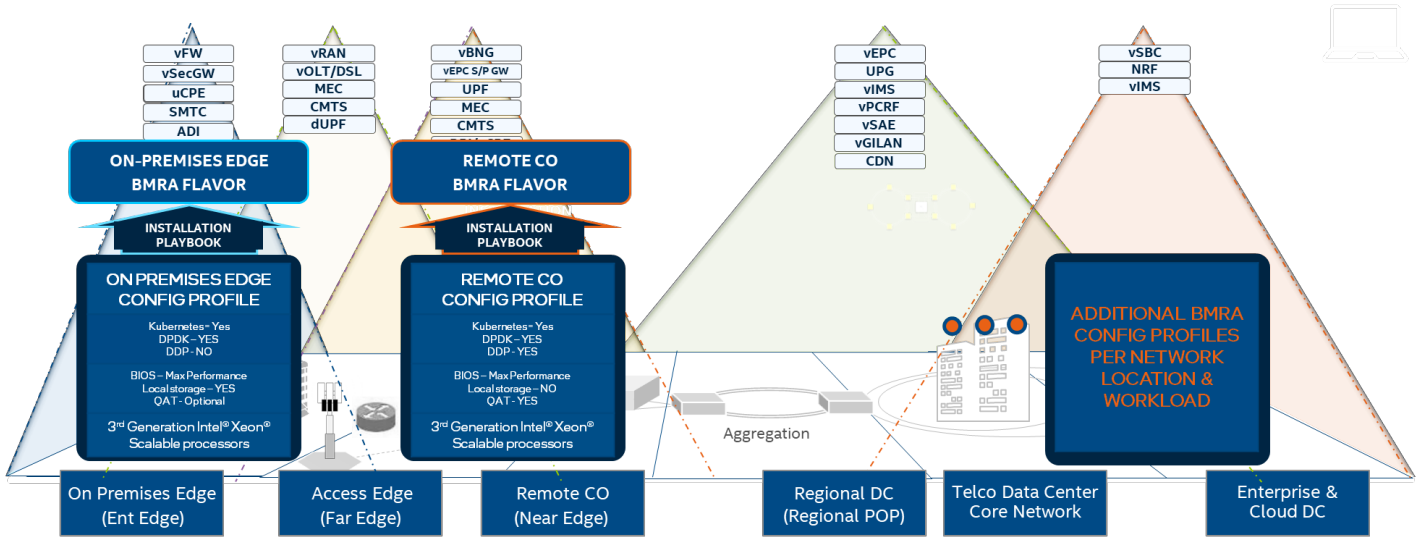

Figure 4. Example of Container Bare Metal Configuration Profiles Per Location

2.3 Use Cases by Network Location

This section explains the corresponding use cases per network location (see Figure 3). BMRA release 21.09 supports three of the Network Locations shown in Figure 3 (see Section 2.2).

On-Premises Edge (Enterprise Edge)

Small cluster of stationary or mobile server platforms, ranging between one and four servers. Usage scenarios include data collection from sensors, local (edge) processing, and upstream data transmission. Sample locations are hospitals, factory floors, law enforcement, media, cargo transportation, power utilities.

- Optimized data processing (to gain insights and reduce upstream transmission volumes) is priority.
- Power efficiency is important in most use cases.
- Automated infrastructure orchestration capabilities are mandatory.

Remote Central Office (Near Edge)

The near edge consists of clusters ranging from a half rack to a few racks of servers, typically in a pre-existing, repurposed, unmanned structure. The usage scenarios include running latency-sensitive applications near the user (for example, real-time gaming, stock trading, video conferencing).

- Power efficiency is a requirement.
- Resource usage efficiency and sharing are a priority.
- Scaling and redundancy/reliability through scale-out; performance is defined at the cluster level.
- Multitenancy support may be required.
- Operational automation is important (site is unmanned).

Regional Data Center (Regional POP)

The Regional Data Center consists of a management domain with many racks of servers, typically managed and orchestrated by a single instance of resource orchestration. Usage scenarios include services such as content delivery, media, mobile connectivity, and cloud services.

- Automation is mandatory due to scale.
- High connectivity (in the aggregate - not individual data plane performance) is important.
- Scaling and redundancy/reliability through scale-out.
- Multitenancy support may be required.

Telco Data Center Core Network

The Telco Data Center Core Network shares many of the characteristics and usage scenarios of the CO Regional Cloud, although at an even larger scale. The core network runs many of the same workloads as the CO Regional Cloud, with the addition of centralized corporate services (for example, finance, OSS/BSS, multi-cloud service orchestration).

2.4 Configuration Profile Installation Playbooks

Intel provides a set of Ansible scripts and Helm charts that enable easy and fast automatic installation[2] 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 that runs on top of Kubernetes to automate the installation process of plugins and K8s capabilities. BMRA Ansible playbooks enable users to customize multiple parameters to fit their installation requirements.

The BMRA Ansible playbooks take into consideration the Intel BKC (Best Known Configuration) for optimized performance.

Installation is done using one of the top-level BMRA Ansible playbooks and includes three phases:

1. Infrastructure setup, which addresses the initial system configuration, including telemetry setup.
2. Kubernetes setup, which deploys Kubernetes capabilities and its add-ons via Kubespray.
3. Installation and configuration of the system capabilities.

The following figure shows an overview of the BMRA Ansible playbooks. For more details, refer to Section 3. In addition, each Configuration Profile has its own playbook and set of scripts, which are described in the Configuration Profile-specific Appendixes.

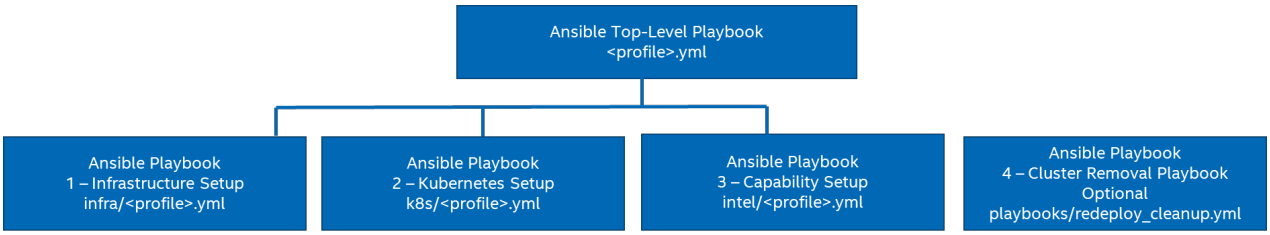

Figure 5. Ansible Playbooks Overview

2.5 Hardware Components

The BMRA supports a range of hardware that enables the different deployment models as explained earlier. Multiple platform hardware options are available, including a variety of 2nd Generation Intel® Xeon® Scalable processor SKUs, 3rd Generation Intel® Xeon® Scalable processor SKUs, Intel® Ethernet Network Adapters, Intel® QAT, and Intel® Server GPU. For details, see Section 5, Reference Architecture Hardware Components and BIOS.

[2] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

2.6 Software Capabilities

The technology described in this document consists of capabilities implemented across the software stack, which enables a Kubernetes environment that uses 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.

2.6.1 Kubernetes Features

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 a 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.

- **Multus** 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.
- **Node Feature Discovery** (NFD) enables generic hardware capability discovery in Kubernetes, including Intel® Xeon® processor-based hardware, for example.
- **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 policy-based decisions. TAS enables automated actions driven by informed decisions based on up-to-date platform telemetry.
- **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 (starting 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.
- **Device plugins**, including Intel® QuickAssist Technology, and SR-IOV device plugins, boost performance and platform efficiency.
- **Kubernetes Operators** are software extensions to Kubernetes that use custom resources to manage applications and their components. Operators follow Kubernetes principles for resource automation, enabling automated installation.

2.6.2 Platform System Features

In conjunction with the Kubernetes capabilities mentioned above, Intel is constantly developing new Intel® architecture platform-level system capabilities for enhanced and deterministic application and network performance.

- **Dynamic Device Personalization (DDP)** is one of the key technologies of the Intel® Ethernet 700 and 800 Series. It enables workload-specific optimizations using the programmable packet processing pipeline to support a broader range of traffic types.
- **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.
- **Intel® Advanced Vector Extensions 512 (Intel® AVX-512)** promotes 512-bit SIMD instruction extensions. They include extensions of the Intel AVX family of SIMD instructions but are encoded using a new encoding scheme. This scheme supports 512-bit vector registers (up to 32 vector registers in 64-bit mode) and conditional processing using opmask registers.
- **Intel® Speed Select Technology – Base Frequency (Intel® SST-BF)** offers a deterministic higher-frequency pool of processing cores to execute high priority workloads and a pool of cores running at lower frequency for non-critical workloads.
- **Intel® Speed Select Technology – Core Power (Intel® SST-CP)** allows OS/VMM to control which cores can take advantage of any power headroom that is available in the system to enable greater system performance.
- **Intel® Speed Select Technology – Performance Profile (Intel® SST-PP)** unlocks greater flexibility for defining core throughput and determinism by allowing greater granularity of core performance and a more effective balance between core count, thermals, and frequency.
- **Intel® Speed Select Technology – Turbo Frequency (Intel® SST-TF)** provides an ability to select a pool of cores to opportunistically access additional Turbo Mode frequencies (P0) for high throughput workloads.
- **Secure Cloud Native Network Platforms** – Using Intel® Security Libraries for Data Center (Intel® SecL – DC), BMRA can help you to build end-to-end platform security. Intel® SecL - DC integrates platform attestation into the cloud native architecture and uses Kubernetes to orchestrate and run workloads only on trusted pods.
- **Key Management Reference Application (KMRA) with Intel® Software Guard Extensions (Intel® SGX)** demonstrates the integration of the asymmetric key capability of Intel® SGX with a third-party hardware security model (HSM) on a centralized key server.

- **Intel® QAT Engine for OpenSSL** is a new option to the libcrypto general purpose cryptographic library. This implementation supports secure applications by directing the requested cryptographic operations to the hardware or software capability present on the underlying platform.
- **Media** processing on the Intel® Server GPU card. The Intel® Server GPU is the first discrete graphics processing unit for data centers based on the new Intel Xᵉ architecture, with support for MPEG-2, AVC, HEVC, and VP9 transcoding plus AV1 decode.

2.6.3 Observability

This solution deploys a broad range of telemetry collectors. Platform collectors capture metrics from CPU performance monitoring, which includes Intel hardware features such as RAS, power, PMU, RDT, NVME storage, network interfaces, disks, platform power, memory, and thermal. In addition to hardware telemetry, BMRA captures DPDK and OVS metrics.

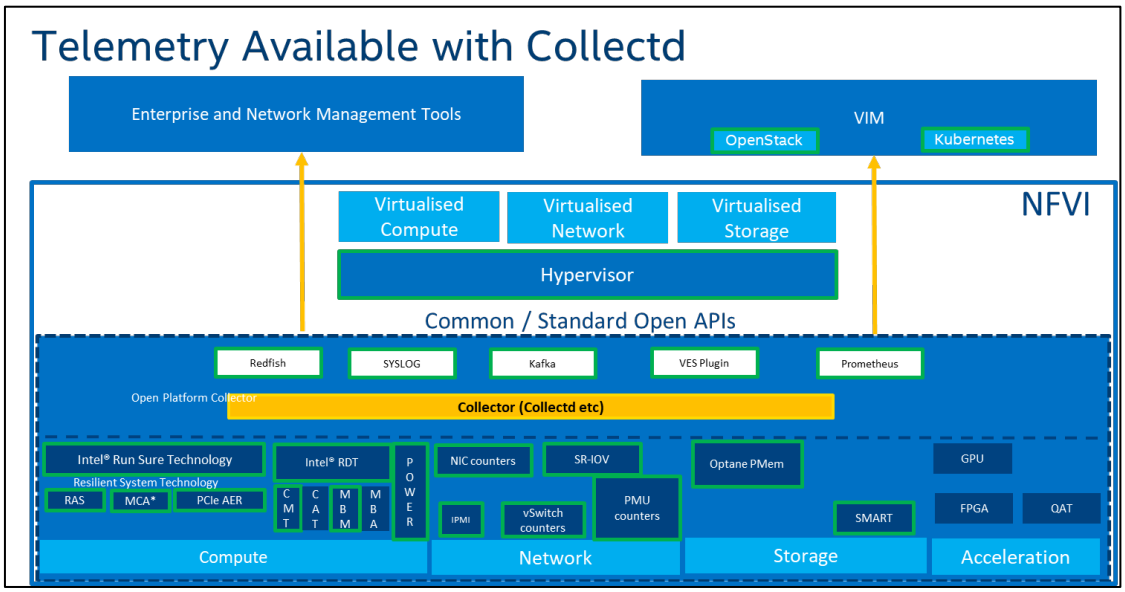

Figure 6. Platform Telemetry Available with Collectd[3]

Metrics can be published to Prometheus and other interfaces such as SNMP, Kafka, and VES.

The telemetry software stack consists of the following components:

- **Collectd** – UNIX daemon that collects a wide range of platform telemetry, including RDT and PMU metrics
- **Telegraf** – Cloud native server telemetry collection agent that collects a wide array of platform and application telemetry, including RDT, PMU, and DPDK metrics
- **Node Exporter** – Platform telemetry monitoring agent
- **Prometheus** – Telemetry monitoring system with time series database
- **Grafana** – Telemetry visualization application
- **Prometheus Adapter** - Implementation of the Kubernetes resource metrics API and custom metrics API. Enables Telemetry Aware Scheduler
- **Telemetry Aware Scheduler** - Makes telemetry data available to scheduling and descheduling decisions in Kubernetes
- **Intel Telemetry Insight Reports** - software available under NDA

Telemetry Insight Report software available from Intel® translates raw platform metrics into up-leveled networking and operational insights, providing insights on platform reliability, utilization, congestion, and configuration issues. These insights can be used to notify NetOps and provide key inputs for remediation actions by automated control systems as part of an observability solution in closed loop systems. To offer meaningful insights from this information, Intel has created a portfolio of telemetry reports that provides actionable data about the current operational status of the server, whether it is overloaded/congested, misconfigured, or in an unhealthy state. Telemetry reports are available under NDA on the Intel Resource & Design Center at the following link: https://www.intel.com/content/www/us/en/secure/design/internal/content-details.html?DocID=645751. Contact your Intel representative for information.

[3] Refer to https://software.intel.com/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products. See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

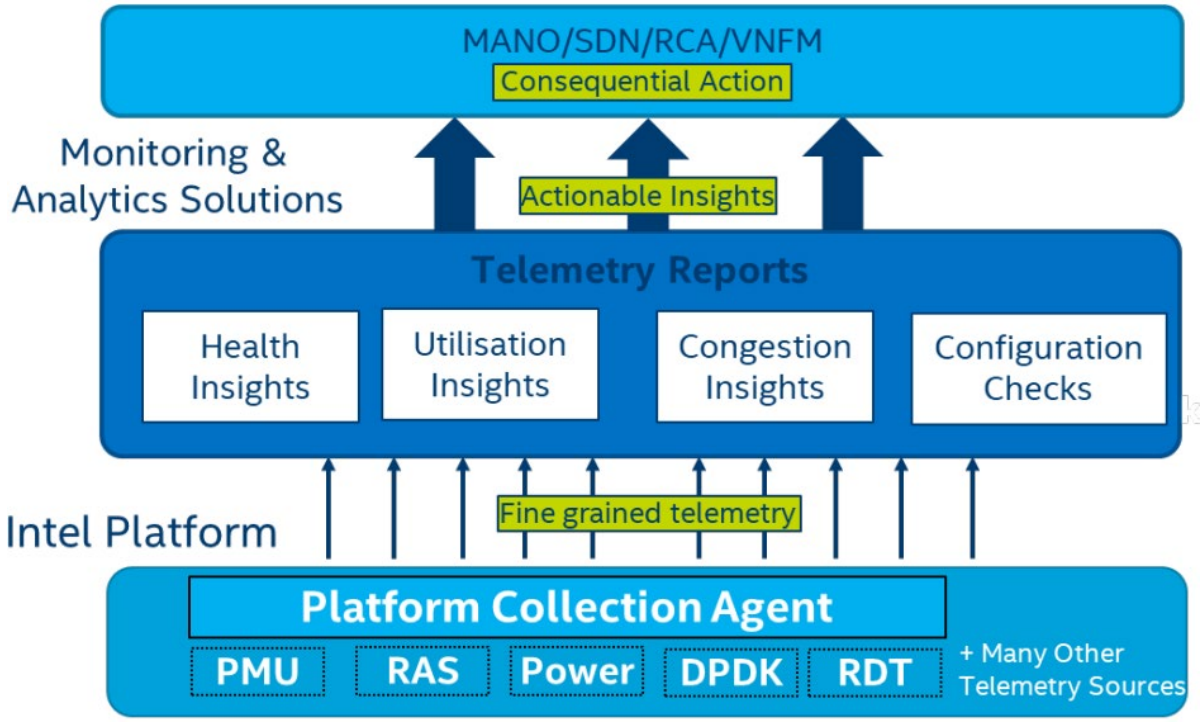

Figure 7. Telemetry Insight Reports High-Level Architecture

Part 2:

Reference Architecture Deployment: Ansible Playbooks Common Hardware Components Software Ingredients Recommended Configurations

3 Reference Architecture Deployment – Ansible Playbooks

This chapter explains how a BMRA Flavor is generated and deployed. The process includes installation of the hardware setup followed by system provisioning using Ansible playbook. The chapter contains the following sections:

- Reference Architecture Installation Prerequisites
- Ansible Playbook Review, including description of building blocks and phases
- Deployment using Ansible Playbook

Note: Ansible playbooks for 2nd Generation Intel Xeon Scalable processors and 3rd Generation Intel Xeon Scalable processors are open source and available here.

3.1 Reference Architecture Installation Prerequisites

Before the Ansible playbook can begin, you must identify the required hardware components, hardware connectivity, and complete the initial configuration, for example BIOS setup. This section helps you get ready for your Ansible installation.

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

- Hardware BOM selection and setup
- 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 for 2nd Generation Intel Xeon Scalable processors and 3rd Generation Intel Xeon Scalable processors can be downloaded directly from the dedicated GitHub page (https://github.com/intel/container-experience-kits/releases) or cloned using the Git.

3.1.1 Hardware BOM Selection and Setup for Control and Worker Nodes

Before software deployment and configuration of BMRA, administrators must deploy the physical hardware infrastructure for their site. To obtain ideal performance and latency characteristics for a given network location, Intel recommends the following hardware configurations:

- Control Nodes - Review Section 5.1 for recommended Control node assembly.
- Worker Nodes – Refer to the following sections for recommended Worker node assembly:
  - Base Worker Node – Review Section 5.2 to satisfy base performance characteristics.
  - Plus Worker Node – Review Section 5.3 to satisfy plus performance characteristics.

Appendix B.1, Appendix C.1, Appendix D.1, Appendix E.1, and Appendix F.1 contain details about the hardware BOM selection and setup.

3.1.2 BIOS Selection for Control and Worker Nodes

Enter the UEFI or BIOS menu and update the configuration as listed in Appendix A.1 and Table 18, which describe the BIOS selection in detail.

3.1.3 Operating System Selection for Control and Worker Nodes

The following Linux operating systems are supported for Control and Worker Nodes:

- CentOS Linux Version 8 (8.3)
- RHEL for x86_64 Version 8 (8.3, 8.4)
- Ubuntu 20.04 LTS (20.04.2)
- Ubuntu 21.04 (21.04.1)

For all supported distributions, the base images are sufficient to be built using the "Minimal" option during installation. In addition, the following must be met:

- The Control and Worker Nodes must have network connectivity to the Ansible Host.
- SSH connections are supported. If needed, on Ubuntu, install SSH Server with the following commands (internet access is required):

```
# sudo apt update
# sudo apt install openssh-server
```

3.1.4 Network Interface Requirements for Control and Worker Nodes

The following list provides a brief description of different networks and network interfaces needed for deployment. (see Figure 1)

- 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 Calico pod network interface (This can be a shared interface with the internet network)
  - Kubernetes control and worker node inter-node communications
  - Calico pod network runs over this network
  - Configured to use a private static address
- Tenant data networks
  - Dedicated networks for traffic
  - SR-IOV enabled
  - VF can be DPDK bound in pod

3.1.5 Software Prerequisites for Ansible Host, Control Nodes, and Worker Nodes

Before deployment of the BMRA Ansible playbooks, the Ansible Host must be prepared. To successfully run the deployment, perform the following tasks before you download the BMRA Ansible code.

Perform the following steps:

1. Log in to the Ansible host machine using SSH or your preferred method to access the shell on that machine.
2. Install packages on Ansible Host. The following example assumes that the host is running CentOS 8.2. Other operating systems may have slightly different installation steps:

```
$ yum install python3
$ pip3 install ansible==2.9.20
$ pip3 install jinja2 -upgrade
$ yum install libselinux-python3
```

3. Enable passwordless login between all nodes in the cluster.
   Create authentication SSH-Keygen keys on Ansible Host:

```
$ ssh-keygen
```

   Upload generated public keys to all the nodes from Ansible Host:

```
$ ssh-copy-id root@<target_server_address>
```

3.2 Ansible Playbook Review

The reference architecture is configured and provisioned automatically using Ansible scripts[4]. Each Configuration Profile has a dedicated Ansible playbook. This section describes how the Ansible playbooks allow for an automated deployment of a fully functional BMRA cluster, including initial system configuration, Kubernetes deployment, and setup of capabilities as described in Section 3.3.

3.2.1 Ansible Playbooks Building Blocks

The following components make up the BMRA Ansible playbooks.

Configuration Files provide examples of cluster-wide and host-specific configuration options for each of the Configuration Profiles. With minimal changes they can be used directly with their corresponding playbooks. For default values refer to the Configuration Profile-specific sections for configuration options for your desired Configuration Profile: : Appendix B BMRA Basic Configuration Profile Setup, Appendix C BMRA Full Configuration Profile Setup, Appendix D BMRA On-Premises Edge Configuration Profile Setup, Appendix E BMRA Remote CO-Forwarding Configuration Profile Setup, Appendix F BMRA Regional Data Center Configuration Profile Setup.

- `inventory.ini`
- `group_vars`
- `host_vars`

Ansible Playbooks act as a user entry point and include all relevant Ansible roles and Helm charts. Top-level Ansible playbooks exist for each Configuration Profile, which allows lean use case-oriented cluster deployments. Each playbook includes only the Ansible roles and configuration files that are relevant for a given use case. See High Level Ansible Playbooks in Figure 8.

- `basic.yml`
- `full_nfv.yml`
- `on_prem.yml`

[4] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

- `remote_fp.yml`
- `regional_dc.yml`

Additionally, a Cluster Removal Playbook exists to optionally remove an existing cluster in case user wants to try different deployment models.

- redeploy_cleanup.yml

Each of these playbooks encompasses **Ansible Roles** grouped into three main execution phases, which are depicted in Figure 8 and further explained in the next section:

- Infrastructure Setup
- Kubernetes Deployment
- Capabilities Setup

Note that several Capabilities Setup roles include nested Helm charts for easier deployment and lifecycle management of deployed applications as well as a group of **Common Utility Roles** that provide reusable functionality across the playbooks.

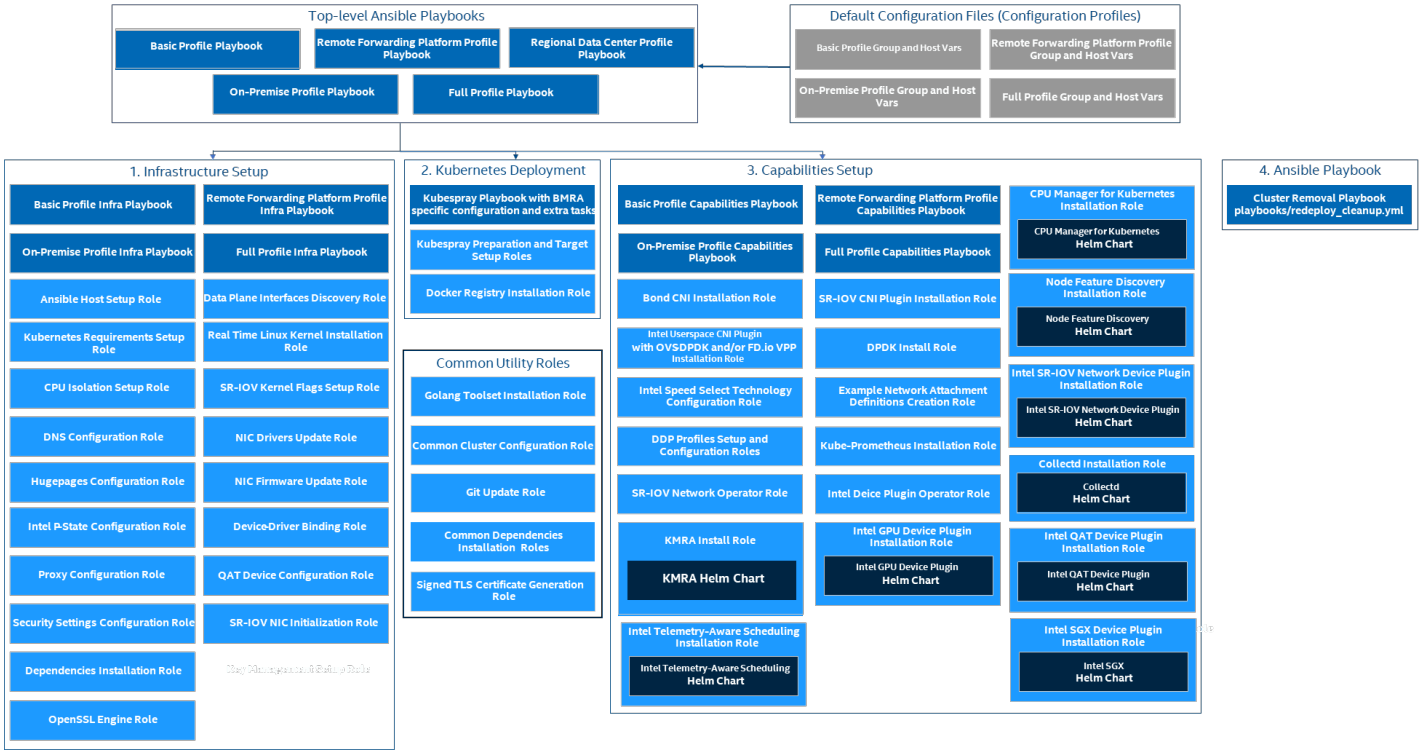

Figure 8. High Level BMRA Ansible Playbooks Architecture[5]

3.2.2 Ansible Playbook Phases

Regardless of the selected Configuration Profile, the installation process always consists of three main phases:

1. **Infrastructure Setup** (sub-playbooks located in playbooks/infra/ directory)
   These playbooks modify kernel boot parameters and apply the initial system configuration for the cluster nodes. Depending on the selected Configuration Profile this includes:
   - Generic host OS preparation, e.g., installation of required packages, Linux kernel configuration, proxy and DNS configuration, and modification of SELinux policies and firewall rules.
   - Configuration of the kernel boot parameters according to the user-provided configuration in order to configure CPU isolation, SR-IOV related settings such as IOMMU, hugepages, or explicitly enable/disable Intel P-state technology.
   - Configuration of SR-IOV capable network cards and QAT devices. This includes the creation of Virtual Functions and binding to appropriate Linux kernel modules.
   - Network Adapter drivers and firmware updates, which help ensure that all latest capabilities such as DDP profiles are enabled.
   - Intel® Speed Select Technology (Intel® SST) configuration, which provides control over base frequency.
   - Installation of Dynamic Device Personalization profiles, which can increase packet throughput, help reduce latency, and lower CPU usage by offloading packet classification and load balancing to the network adapter.

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

2. **Kubernetes Setup** (located in playbooks/k8s/ directory)
   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 Calico and Helm. Preferred security practices are used in the default configuration. On top of Kubespray, there's also a container registry instance deployed to store images of various control-plane Kubernetes applications such as TAS, CMK, or device plugins.
3. **BMRA System Capabilities Setup** (sub-playbooks located in playbooks/intel directory):
   Advanced networking technologies, Enhanced Platform Awareness, and device plugin features are deployed by this playbook using Operators or Helm Charts as part of the BM RA. The following capabilities are deployed:
   - Device plugins that allow using as an example SR-IOV, QAT, and GPU devices in workloads running on top of Kubernetes.
   - SR-IOV CNI plugin, Bond CNI plugin, and Userspace CNI plugin, which allow Kubernetes pods to be attached directly to accelerated and highly available hardware and software network interfaces.
   - 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.
   - Telemetry Aware Scheduling, which allows scheduling workloads based on telemetry data.
   - Full Telemetry Stack consisting of Collectd, Kube-Prometheus, and Grafana, which gives cluster and workload monitoring capabilities and acts as a source of metrics that can be used in TAS to orchestrate scheduling decisions.

3.3 Deployment using Ansible Playbook

This section describes common steps that need to be executed in order to obtain the BMRA Ansible Playbooks source code, prepare target servers, configure inventory and variable files, and deploy the BMRA Kubernetes cluster.

3.3.1 Prepare Target Servers

For each target server that will act as a control or worker node, you must make sure that it meets the following requirements:

- Python 3 is installed. The version depends on the target distribution.
- SSH keys are exchanged between the Ansible host and each target node. The same SSH keys need to be configured on each of the machines. You can achieve that by executing below command on the Ansible host to copy to each target server in the cluster:
  ```
  ssh-copy-id root@<target_server_address>
  ```
- Internet access on all target servers is mandatory. Proxies are supported and can be configured in the Ansible vars.
- BIOS configuration matching the desired state is applied. For details, refer to the specific Configuration Profile Appendix for your profile: Appendix B BMRA Basic Configuration Profile Setup, Appendix C BMRA Full Configuration Profile Setup, Appendix D BMRA On-Premises Edge Configuration Profile Setup, Appendix E BMRA Remote CO-Forwarding Configuration Profile Setup, Appendix F BMRA Regional Data Center Configuration Profile Setup.

For detailed steps on how to build the Ansible host, refer to Appendix A.1.

3.3.2 Get Ansible Playbook and Prepare Configuration Templates

Perform the following steps:

1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks – using the tag from Table 22, for example:
   ```
   git checkout v21.09
   ```
   Note: Alternatively go to https://github.com/intel/container-experience-kits/releases, download the latest release tarball, and unarchive it:
   ```
   wget https://github.com/intel/container-experience-kits/archive/v21.09.tar.gz
   tar xf v21.09.tar.gz
   cd container-experience-kits-21.09
   ```
4. Initialize Git submodules to download Kubespray code:
   ```
   git submodule update --init
   ```
3. Decide which Configuration Profile you want to use and export the environmental variable.
   For Kubernetes **Basic** Configuration Profile deployment:
   ```
   export PROFILE=basic
   ```
   For Kubernetes **Full** Configuration Profile deployment:
   ```
   export PROFILE=full_nfv
   ```
   For Kubernetes **On-Premises Edge** Configuration Profile deployment:

```
export PROFILE=on_prem
```

For Kubernetes **Remote Central Office-Packet Forwarding** Configuration Profile deployment:

```
export PROFILE=remote_fp
```

For Kubernetes **Regional Data Center** Configuration Profile deployment:

```
export PROFILE=regional_dc
```

4. Install requirements needed by `render.py` script:

```
pip3 install -r profiles/requirements.txt
```

5. Generate example profiles:

```
make bmra-profiles
```

6. Copy example inventory file to the project root dir:

```
cp examples/${PROFILE}/inventory.ini .
```

5. Copy example configuration files to the project root dir:

```
cp -r examples/${PROFILE}/group_vars examples/${PROFILE}/host_vars .
```

3.3.3 Update Ansible Inventory File

Perform the following steps:

1. Edit the `inventory.ini` file copied in the previous steps.
2. In the section `[all]`, specify all your target servers. Use their actual hostnames and Management IP addresses. Also list your Ansible host as "`localhost`" with the Python version packaged with your OS distribution.

   Example: There is an Ansible host and three servers - one controller and two worker nodes. Their hostnames are `controller1`, `node1`, and `node2` respectively. They are all located in the 10.100.200.0/24 subnet and are assigned static IP addresses (10.100.200.10x). All three cluster servers have CentOS 8 installed with Python 3, and the Ansible host has Ubuntu 20.04 with Python 3. In this case the `[all]` section should be configured like this:

```
[all]
localhost ansible_python_interpreter=/usr/bin/python3
controller1 ansible_host=10.100.200.101 ip=10.100.200.101
node1       ansible_host=10.100.200.102 ip=10.100.200.102
node2       ansible_host=10.100.200.103 ip=10.100.200.103

...

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

3. Assign servers to Ansible groups.

   Example: As mentioned above, we want to have one controller and two worker nodes. It's recommended to place "`etcd`" (cluster database) on the same node as the remaining control-plane components. So, in that case, the final `inventory.ini` file would look like this:

```
[all]
localhost ansible_python_interpreter=/usr/bin/python3
controller1 ansible_host=10.100.200.101 ip=10.100.200.101
node1       ansible_host=10.100.200.102 ip=10.100.200.102
node2       ansible_host=10.100.200.103 ip=10.100.200.103


[kube_control_plane]
controller1

[etcd]
controller1

[kube_node]
node1
node2

[k8s_cluster:children]
kube_control_plane
kube_node

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

3.3.4 Update Ansible Host and Group Variables

Perform the following steps:

1. Create `host_vars` files for all nodes, matching their hostnames from the inventory file.

   Example: Following the setup from previous steps, we need a file for each of the nodes. We do not need a host vars file for the controller:
   ```
   cp host_vars/node1.yml host_vars/node2.yml
   ```
2. Edit `host_vars/<node_name>.yml` and `group_vars/all.yml` files to match your desired configuration. Each Configuration Profile uses its own set of variables. Refer to the specific Configuration Profile Appendix for your profile to get a full list of variables and their documentation: Appendix B BMRA Basic Configuration Profile Setup, Appendix C BMRA Full Configuration Profile Setup, Appendix D BMRA On-Premises Edge Configuration Profile Setup, Appendix E BMRA Remote CO-Forwarding Configuration Profile Setup, Appendix F BMRA Regional Data Center Configuration Profile Setup.

3.3.5 Run Ansible Cluster Deployment Playbook

After the inventory and vars are configured, you can run the provided playbooks from the root directory of the project.

It is recommended that you check dependencies of components enabled in `group_vars` and `host_vars` with the packaged dependency checker:
```
ansible-playbook -i inventory.ini playbooks/preflight.yml
```

If you are deploying an RHEL 8 cluster you need to patch kubespray:
```
ansible-playbook -i inventory.ini playbooks/k8s/patch_kubespray.yml
```

Otherwise, you can skip directly to your chosen Configuration Profile playbook:
```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

Pay attention to logs and messages displayed on the screen. Depending on the selected Configuration Profile, network bandwidth, storage speed, and other similar factors, the execution may take up to 30-40 minutes.

After the playbook finishes without any “Failed” tasks, you can proceed with the deployment validation described in Section 7, Post Deployment Verification Guidelines.

Note: Additional information can be found in the Ansible Playbook readme.

3.3.6 Run Ansible Cluster Removal Playbook

If the playbook fails or if you want to clean up the environment to run a new deployment, you can optionally use the provided Cluster Removal Playbook (`redeploy_cleanup.yml`) to remove any previously installed Kubernetes and related plugins.
```
ansible-playbook -i inventory.ini playbooks/redeploy_cleanup.yml
```

After successful removal of Kubernetes components, you can repeat Section 3.3.5.

Note: Any OS and/or hardware configurations (for example, proxies, drivers, kernel parameters) are not reset by the cleanup playbook.

4 Software Capabilities Review

Intel, in collaboration with industry partners, continues to work to bring Intel® architecture advancements to the cloud native ecosystem through support of **Kubernetes features**, extension of **Kubernetes plugins**, and development of **system hardware resources** as described below[6].

4.1 Container Runtimes

A container runtime is software that executes containers and manages container images on a node. The following container runtimes are available in BMRA.

4.1.1 Docker

Docker is container platform comprised of the following elements.

- The Docker daemon, named `dockerd`, that manages Docker containers and handles container objects; the daemon listens for request via the Docker Engine API
- The Docker client, which is the primary way of interacting with Docker
- Docker registries
- Docker objects

For more details, see: https://www.docker.com/

[6] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

4.1.2 Containerd

Containerd is an Open Container Initiative (OCI)-compliant core container runtime that implements the Kubernetes Container Runtime Interface (CRI) via CRI plugin. Most interactions are handled via `runc` and/or OS-specific libraries.

For more details, see: https://containerd.io/

4.1.3 CRI-O

CRI-O is an implementation of the Kubernetes CRI that enables using OCI-compatible runtimes. It allows Kubernetes to use any OCI-compliant runtime as the container runtime for running pods. It supports `runc` and Kata Containers as the container runtimes, but any OCI-conformant runtime can be used.

For more details, see: https://cri-o.io/

4.2 Kubernetes Plugins

The following device plugins are used to advertise Intel® architecture system hardware resources to Kubernetes. Several of the plugins are Container Network Interface (CNI), which is explained here: https://github.com/containernetworking/cni

4.2.1 Multus CNI

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.

Supporting multiple network interfaces is a key requirement for many network functions (NFs), 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. A set of basic plugins is provided through CNI Plugins from the container networking team, and the below plugins, SR-IOV, Userspace and Bond CNIs, can be installed as part of the deployment using Ansible playbook.

For more details see: https://github.com/intel/multus-cni

4.2.2 SR-IOV Network Device Plugin

Single Root Input/Output Virtualization (SR-IOV) provides I/O virtualization that makes a single PCIe device (typically a network adapter) appear as many network devices, also known as virtual functions (VFs) in the Linux kernel. In Kubernetes, this results in network connections that can be separately managed and assigned to different pods.

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.

The SR-IOV network device plugin provides a way to filter the available VFs and make them available as endpoints in Kubernetes. Using a list of selectors, a subset of VFs can be associated with a resource name that can be used when assigning resources to pods. An example of a resource is shown below:

```
"resourceName": "intel_sriov_netdevice",
"selectors": {
    "vendors": ["8086"],
    "devices": ["154c", "10ed", "1889"],
    "drivers": ["iavf", "i40evf", "ixgbevf"]
}
```

For more details, see: https://github.com/intel/sriov-network-device-plugin

4.2.3 SR-IOV CNI

The Single Root I/O Virtualization SR-IOV device plugin enables partition of a single physical network adapter (PCI) resource into virtual PCI functions (VFs) that can be attached to Kubernetes pods. To attach an SR-IOV device resource to the Kubernetes pod network, we use the SR-IOV CNI. The SR-IOV CNI plugin enables the 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.

For more details, see: https://github.com/intel/sriov-cni

4.2.4 Userspace CNI

The Userspace CNI is a Container Network Interface (CNI) designed to implement userspace networking (as opposed to kernel space networking), such as DPDK-based applications. It is designed to run with either OVS-DPDK or VPP along with the Multus CNI plugin in Kubernetes deployments. Userspace CNI provides a high-performance container networking solution and data plane acceleration for containers.

4.2.5 Bond CNI

Bond CNI allows for aggregation of multiple network interfaces into one logical interface. Interface bonding is used to provide additional network capacity and/or redundancy. When used as standalone plugin, interfaces are obtained from the host's network namespace. Bonded interface is created in the container network namespace. When used with Multus you can bond interfaces that were previously passed to the container.

4.2.6 Intel® QuickAssist Device Plugin

Intel® QuickAssist Adapters integrate hardware acceleration of compute-intensive workloads, such as bulk cryptography, public key exchange, and compression, on Intel® architecture platforms. The Intel® QAT device plugin for Kubernetes supports Intel® QuickAssist Adapters and includes an example scenario that uses the Data Plane Development Kit (DPDK) drivers.

For more details, see: https://github.com/intel/intel-device-plugins-for-kubernetes

4.2.7 Intel® Software Guard Extensions (Intel® SGX) Device Plugin

Intel® SGX device plugin allows Kubernetes workloads to use Intel® SGX on 3rd Generation Intel® Xeon® Scalable processors. The plugin is used in conjunction with an SGX Admission webhook and EPC memory registration to isolate specific application code and data in memory via enclaves that are designed to be protected from processes running at higher privilege levels. An additional remote attestation server is required to verify software is running inside an SGX enclave on a trusted computing node.

For more details, see: https://github.com/intel/intel-device-plugins-for-kubernetes

4.3 Kubernetes Features

Kubernetes (K8s) is a leading open-source orchestration platform 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[7]. This section outlines K8s capabilities common to all profiles within the BMRA 21.09.

4.3.1 Node Feature Discovery

In a standard deployment, Kubernetes reveals very few details about the underlying platform to the user. This may be a good strategy for general data center use, but, in many cases a workload behavior or its performance may improve by leveraging the platform (hardware and/or software) features. Node Feature Discovery (NFD) is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform that can be used to facilitate intelligent scheduling of a workload. NFD currently detects following features:

- CPUID: NFD advertises CPU features such as Intel® Advanced Vector Extensions (Intel® AVX). Certain workloads, such as machine learning, may gain a significant performance improvement from these extensions.
- SR-IOV networking: NFD detects the presence of SR-IOV-enabled NICs, allowing optimized scheduling of network-intensive workloads.
- Intel® Resource Director Technology (Intel® RDT): Intel RDT allows visibility and control over the use of last-level cache (LLC) and memory bandwidth between co-running workloads. By allowing allocation and isolation of these shared resources, and thus reducing contention, Intel RDT helps in mitigating the effects of noisy neighbors. NFD detects the different Intel RDT technologies supported by the underlying hardware platform.
- Intel® Turbo Boost Technology: NFD detects the state of Intel® Turbo Boost Technology, allowing optimal scheduling of workloads that have a well-understood dependency on this technology.
- IOMMU: An input/output memory management unit (IOMMU), such as Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d) technology, allows isolation and restriction of device accesses. This enables direct hardware access in virtualized environments, highly accelerating I/O performance by removing the need for device emulation and bounce buffers.
- SSD storage: NFD detects the presence of non-rotational block storage on the node, making it possible to accelerate workloads requiring fast local disk access.
- NUMA topology: NFD detects the presence of NUMA topology, making it possible to optimize scheduling of applications based on their NUMA-awareness.

[7] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

- Linux kernel: NFD detects the kernel version and advertises it through multiple labels, allowing the deployment of workloads with different granularity of kernel version dependency.
- PCI: NFD detects PCI devices, allowing optimized scheduling of workloads dependent on certain PCI devices.

Some of the features that NFD currently detects are shown in the following table. For more information, see https://kubernetes-sigs.github.io/node-feature-discovery/v0.9/get-started/features.html#feature-sources.

Table 4. Features Detected by NFD[8]

FEATURE SOURCE	FEATURE NAME	ATTRIBUTE	DESCRIPTION
cpu	cpuid	<cpuid flag>	CPU capability is supported
	hardware_multithreading		Hardware multithreading, such as Intel® Hyper-Threading Technology (Intel® HT Technology), is enabled
	power	sst_bf.enabled	Intel SST-BF is enabled
	pstate	status	The status of the Intel P-state driver when in use and enabled; either 'active' or 'passive'
		turbo	Set to 'true' if turbo frequencies are enabled in Intel P-state driver; set to 'false' if they have been disabled
		scaling_governor	The value of the Intel P-state scaling_governor when in use; either 'powersave' or 'performance'
	cstate	enabled	Set to 'true' if C-states are set in the intel_idle driver; otherwise set to 'false'. Clear if intel_idle cpuidle driver is not active
	rdt	RDTMON	Intel RDT monitoring
		RDTCMT	Cache Monitoring Technology (CMT)
		RDTMBM	Memory Bandwidth Monitoring (MBM)
		RDTL3CA	L3 Cache Allocation Technology
		RDTL2CA	L2 Cache Allocation Technology
		RDTMBA	Memory Bandwidth Allocation (MBA)
iommu	enabled		IOMMU is present and enabled in the kernel
kernel	config	<option name>	Kernel config option is enabled (set 'y' or 'm'). Default options are NO_HZ, NO_HZ_IDLE, NO_HZ_FULL, and PREEMPT
	selinux	enabled	SELinux is enabled on the node
	version	full	Full kernel version as reported by `/proc/sys/kernel/osrelease` (for example, '4.5.6-7-g123abcde')
		major	First component of the kernel version (for example, '4')
		minor	Second component of the kernel version (for example, '5')
		revision	Third component of the kernel version (for example, '6')
memory	numa		Multiple memory nodes, for example, NUMA architecture detected
	nv	present	NVDIMM devices are present
	nv	dax	NVDIMM regions configured in DAX mode are present
network	sriov	capable	SR-IOV-enabled network interface cards are present
		configured	SR-IOV virtual functions have been configured
pci	<device label>	present	PCI device is detected
	<device label>	sriov.capable	SR-IOV-enabled PCI device present
storage	nonrotationaldisk		Non-rotational disk, such as SSD, is present in the node
system	os_release	ID	Operating system identifier
		VERSION_ID	Operating system version identifier (for example, '6.7')
		VERSION_ID.major	First component of the OS version id (for example, '6')
		VERSION_ID.minor	Second component of the OS version id (for example, '7')
usb	<device label>	present	USB device is detected

[8] Paraphrased from "Feature Discovery · Node Feature Discovery," · Node Feature Discovery (Kubernetes SIGs, August 20, 2021), https://kubernetes-sigs.github.io/node-feature-discovery/v0.9/get-started/features.html#feature-sources.

For more details see: https://github.com/kubernetes-sigs/node-feature-discovery

4.3.2 Topology Manager

Today's systems use a combination of CPUs and hardware accelerators to support latency-critical execution and high-throughput parallel computation. To help extract the optimal performance out of such systems, the required optimizations related to CPU isolation, memory, and device locality must be made. In Kubernetes however, these optimizations are handled by a disjointed set of components. Topology Manager is a feature of Kubernetes distribution. It is a kubelet component that coordinates the set of components that are responsible for making topology aligned resource allocations.

For details see: https://builders.intel.com/docs/networkbuilders/topology-management-implementation-in-kubernetes-technology-guide.pdf

4.3.3 Kubernetes Native CPU Manager

Native CPU Manager for Kubernetes provides mechanisms for allocation of CPU cores to workloads in situation where pods contend for resources of the CPU. When contention happens workloads may get moved to other CPUs and workload performance may be impacted. To avoid this, CPU Manager offers an option to allocate exclusive cores to a workload (pod) by specifying "guaranteed QoS" and integer CPU requests.

See: https://kubernetes.io/blog/2018/07/24/feature-highlight-cpu-manager/

4.3.4 CPU Manager for Kubernetes (CMK)

Intel's CPU Manager for Kubernetes provides mechanisms for CPU core pinning and isolation of containerized workloads. Intel's CMK divides the CPUs on a system into three pools (by default) with an additional optional pool. The CPUs are divided into pools according to exclusivity: exclusive, shared, and infra. The CMK maintains lists of CPUs in each of the lists from which user containers can acquire CPUs. Intel's CMK delivers predictable performance for high-priority applications due to reduced or eliminated thread preemption and maximizing CPU cache utilization. CMK provides a single multiuse command-line program to perform various functions for host configuration, managing groups of CPUs, and constraining workloads to specific CPUs.

See: https://builders.intel.com/docs/networkbuilders/cpu-pin-and-isolation-in-kubernetes-app-note.pdf

4.3.5 Telemetry Aware Scheduling

Telemetry Aware Scheduling (TAS) makes telemetry data available for scheduling and descheduling decisions in Kubernetes. Through a user-defined 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.

For example, a pod that requires certain cache characteristics can be scheduled based on output from Intel® RDT metrics. Likewise, a combination of RDT, RAS, and other platform metrics could be used to provide a signal for the overall health of a node and could be used to help proactively ensure workload resiliency.

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 these policies 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 three 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 is never scheduled on a node that breaks any one of the 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
  ```

TAS allows arbitrary, user-defined rules to be put in place in order to impact scheduling in a K8s cluster. Using the K8s descheduler, it can evict workloads that are breaking some rules in order to have it replaced on a more suitable node.

In a modern cloud computing cluster, there is a torrent of data that only certain subject matter experts know how to interpret and act upon. In scheduling workloads, operators know on which compute nodes a workload may perform better based on up-to-date utilization metrics. Likewise, certain telemetry values, or combinations of values, can be recognized as signs that a node has some serious problem that is interfering with workload operation. The TAS policy system allows these insights to influence the scheduling and lifecycle placement process – turning implicit personal knowledge into formal, actionable information.

For details, refer to: https://networkbuilders.intel.com/solutionslibrary/telemetry-aware-scheduling-automated-workload-optimization-with-kubernetes-k8s-technology-guide

4.4 Istio Service Mesh

Istio is an open source service mesh that layers transparently onto existing distributed applications. "A service mesh is a dedicated infrastructure layer that you can add to your applications. It allows you to transparently add capabilities like observability, traffic management, and security, without adding them to your own code. The term "service mesh" describes both the type of software you use to implement this pattern, and the security or network domain that is created when you use that software."[9]

Istio's features provide a uniform and efficient way to help secure, connect, and monitor services. Istio is the path to load balancing, service-to-service authentication, and monitoring – with few or no service code changes. Its powerful control plane brings vital features, including:

- More secure service-to-service communication in a cluster with TLS encryption, strong identity-based authentication, and authorization
- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection
- A pluggable policy layer and configuration API supporting access controls, rate limits, and quotas
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress

For more details, refer to: https://istio.io/latest/about/service-mesh/

4.4.1 Istio Deployment Example

You can enable the Intel SGX operator as an external certificate authority to protect the private keys of an Istio service mesh. Istio supports Custom Certificate Authority (CA) integration using Kubernetes Certificate Signing Request (CSR). This requires a Kubernetes version >= 1.18. You can implement Istio with Intel SGX while using the Intel SGX operator as an external CA to Istio, running as a pod in the "sgx-operator" namespace.

Once configured, whenever a new istio-proxy (Envoy container) is injected into a newly started pod, the certificate signing request begins. The istio-proxy auto-generates its own certificate for service mesh communication and requests that it be signed by the Istio custom CA private key. This private key is stored in an encrypted SGX enclave only accessible by the Intel SGX operator. The certificate goes from the Istio operator to the Intel SGX operator, which signs the public certificate within the enclave in DRAM memory. Once signed, it is returned back through the same path to the newly started pod that initiated the request. That pod can now communicate with other pods in the configured service mesh.

4.5 Operators

Kubernetes operators are subject-specific controllers that extend the functionality of the Kubernetes API to create, configure, and manage complex entities on behalf of a Kubernetes user. Kubernetes is designed from the ground up for automation as it includes native mechanisms for deploying, running, and scaling workloads. However, some workloads and services require deeper knowledge of how the system should behave outside the bounds of core Kubernetes. Operators are purpose-built with operational intelligence to address the individuality of such constructs. Tools for operator creation help developers build an automation experience for cluster administrators and end users. By extending a common set of Kubernetes APIs and tools, Kubernetes operators can help provide ease of deployment and streamlined Day 1 and Day 2 operations. The Kubernetes Operator Pattern has emerged as a solution to help ensure the automation of these function-specific applications is possible in a Kubernetes cluster.

4.5.1 SR-IOV Network Operator

The SR-IOV Network Operator is designed to help the user to provision and configure the SR-IOV CNI plugin and device plugin in the Kubernetes cluster.

When SR-IOV Network Operator is enabled by setting `sriov_network_operator_enabled: true` in the `group_vars/all.yml` file, it is mutually exclusive with SR-IOV CNI and device plugins. The plugins must be disabled by setting the

[9] "The Istio Service Mesh." Istio. Accessed October 8, 2021. https://istio.io/latest/about/service-mesh/.

respective options in the `group_vars/host_vars`. SR-IOV Network Operator is enabled by default, and SR-IOV CNI and device plugins are disabled by default.

To configure SR-IOV device custom resource definition (CRD), `SriovNetworkNodePolicy` should be provided.

BMRA handles creation of such CRDs and creates and populates them automatically according to the provided configuration in the `host_vars/node1.yml` file, namely in the section `dataplane_interfaces`.

For more details, refer to: https://github.com/k8snetworkplumbingwg/sriov-network-operator

4.5.2 Intel Device Plugins Operator

Intel device plugins operator is a Kubernetes custom controller. Its goal is to serve the installation and lifecycle management of Intel device plugins for Kubernetes. It provides a single point of control for GPU, QAT, SGX, and FPGA devices to cluster administrators. Device plugins are deployed by applying custom resource, and each device plugin has its own custom resource definition (CRD). The corresponding controller watches CRUD operations to those custom resources. Currently, BMRA uses the Intel device plugins operator to deploy Intel SGX device plugin and Intel GPU device plugin.

For more details, refer to: https://github.com/intel/intel-device-plugins-for-kubernetes

4.5.3 Istio Operator

Istio operator brings the ability to avoid manual install, upgrade, and uninstall of the Istio service mesh. It enables deployment of Istio service mesh by defining the IstioOperator CRD. The Istio operator can be enabled or disabled by setting the `istio_enabled` flag in the `group_vars/all.yml` file. The Istio operator is enabled by default for all BMRA profiles except basic.

BMRA is shipped with a predefined set of Istio configuration profiles. The following Istio configuration profiles are available: default, minimal, external, empty, none. If no Istio configuration profile is defined by the variable `istio_profile` in the `group_vars/all.yml` file, then the default Istio configuration profile is applied. If no Istio configuration profile is set, then no Istio configuration profile is deployed. For more details on Istio configuration profiles, see https://istio.io/latest/docs/setup/additional-setup/config-profiles/. For more details on the Istio operator, see https://istio.io/latest/docs/setup/install/operator.

4.6 Dynamic Device Personalization (DDP)

One of the key technologies of the Intel® Ethernet 700 and 800 Series Network Adapters is Dynamic Device Personalization (DDP). This technology enables workload-specific optimizations using the programmable packet-processing pipeline. Additional protocols in the default set help improve packet processing efficiency and result in enhanced performance in specific use cases.[10]

Additional information about DDP can be found in https://www.intel.com/content/www/us/en/architecture-and-technology/ethernet/dynamic-device-personalization-brief.html

4.6.1 DDP on Intel Ethernet 700 Series Network Adapters

Intel Ethernet 700 Series Network Adapters support only one DDP image loaded on a network card. An image must be loaded to the first physical function of the device (PF0), but the configuration is applied to all ports of the adapter.

For Intel Ethernet 700 Series Network Adapters, BMRA provides the following list of DDP images:

- `ecpri.pkg`
- `esp-ah.pkg`
- `ppp-oe-ol2tpv2.pkgo`
- `mplsogreudp.pkg`
- `gtp.pkgo`

To use specific DDP-enabled virtual functions in BMRA, create a new resource with `ddpImage` selector by extending the `sriovdp_config_data` variable before deployment.

```
PATH: examples/full_nfv/group_vars/all.yml:
sriovdp_config_data: |
    {
        "resourceList": [
            {
                "resourceName": " intel_sriov_dpdk_700_series_gtpgo",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["154c"],
                    "drivers": ["vfio-pci"]
```

[10] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

```
                    "ddpProfiles": ["GTPv1-C/U IPv4/IPv6 payload"]
                }
            },
            {(...)}
        ]
    }
```

For more details about SR-IOV device plugin configuration, refer to https://github.com/intel/sriov-network-device-plugin#configurations

After BMRA Deployment, request the `intel_sriov_dpdk_700_series_gtpgo` resource in the workload's Pod manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net1
spec:
  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_dpdk_700_series_gtpgo: '1'
      limits:
        intel.com/intel_sriov_dpdk_700_series_gtpgo: '1'
```

Keep the following points in mind when working with DDP images on Intel® Ethernet 700 Series Network Adapters:

- DDP profiles are NOT automatically unloaded when the driver is unbound/unloaded. Note that subsequent driver reload may corrupt the profile configuration during its initialization and is NOT recommended.
- DDP profiles should be manually rolled-back before driver unload/unbind if the intention is to start with a clean hardware configuration.
- Exercise caution while loading DDP profiles. Attempting to load files other than DDP profiles provided by Intel may cause system instability, system crashes, or system hangs.

Additional References

- https://software.intel.com/content/www/us/en/develop/articles/dynamic-device-personalization-for-intel-ethernet-700-series.html
- https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-gtpv1-dynamic-device-personalization.pdf
- https://github.com/intel/sriov-network-device-plugin

4.6.2 DDP on Intel® Ethernet 800 Series Network Adapters

Intel® Ethernet 800 Series Network Adapters support a broad range of protocols in DDP profile images[11]. You can choose the default image or the telecommunications (comms) image that supports certain market-specific protocols in addition to protocols in the OS-default package. The OS-default DDP package supports the profiles listed in the following table.

DEFAULT PACKAGE
MAC
EtherType
VLAN
IPv4
IPv6
TCP
ARP
UDP
SCTP
ICMP

[11] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

DEFAULT PACKAGE
ICMPV6
CTRL
LLDP
VXLAN-GPE
VxLAN (non-GPE)
Geneve
GRE
NVGRE
RoCEv2

COMMS PACKAGE
Extends default profile package with the following protocols:
GTP
PPPOE
L2TPv3
IPsec
PFCP

To deploy the communication profile on the network adapter, the host variable `dataplane_interfaces` must contain a proper ddp_profile value. For example:

```
dataplane_interfaces:
  - name: enp24s0f0                # PF interface name
    bus_info: "18:00.0"            # pci bus info
    pf_driver: i40e                # PF driver, "i40e", "ice"
    default_vf_driver: "iavf"      # default driver to be used with VFs
    sriov_numvfs: 6                # total number of VFs to create
    sriov_vfs:                     # list of VFs to create with specific driver
      vf_00: "vfio-pci"            # VF driver to be attached to this VF under this PF.
      vf_05: "vfio-pci"            # VF driver to be attached to this VF under this PF.
    ddp_profile: "ice_comms-1.3.24.0" # DDP package name to be loaded into the NETWORK ADAPTER
```

The image must be loaded to first physical function of the device (PF0), but the configuration is applied to all ports of the adapter.

To deploy a workload with a DDP SR-IOV VF, the `intel_sriov_dpdk_800_series` resource must be requested in the workload's pod manifest, after BMRA deployment.

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net1
spec:
  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_dpdk_800_series: '1'
      limits:
        intel.com/intel_sriov_dpdk_800_series: '1'
```

4.6.2.1 Kubernetes Support for DDP in Intel Ethernet 800-Series Network Adapters

SR-IOV devices (VFs included) are exposed to the Kubernetes pods via SR-IOV device plugin (SR-IOV DP). In BMRA, the plugin cannot discover loaded DDP plugins on Intel Ethernet 800-Series Network Adapters (https://github.com/intel/sriov-network-device-plugin#configurations). As a result, one cannot orchestrate a workload that requires a DDP profile on such network adapters.

However, you can work around this issue in the following ways:
1. You can deploy *comms* profile on every node with Intel Ethernet 800 Series Network Adapter, or
2. You can deploy *comms* profile on a subset of Intel Ethernet 800 Series Network Adapters and label the nodes accordingly using Kubernetes labels. Then deploy the workload that requires specific DDP profile with specific Node Selector.

4.7 Intel® Speed Select Technology

Intel SST is a collection of features that give more granular control over CPU performance.

4.7.1 Intel Speed Select Technology – Base Frequency

Select SKUs of 2nd Generation Intel Xeon Scalable processors (5218N, 6230N, and 6252N) and 3rd Generation Intel Xeon Scalable processors (6318N and 6338N) offer a capability called Intel Speed Select Technology – Base Frequency (Intel SST-BF). Intel SST-BF controls the core frequency model. When enabled, some cores are at higher frequency and the remaining cores run at lower frequency[12].

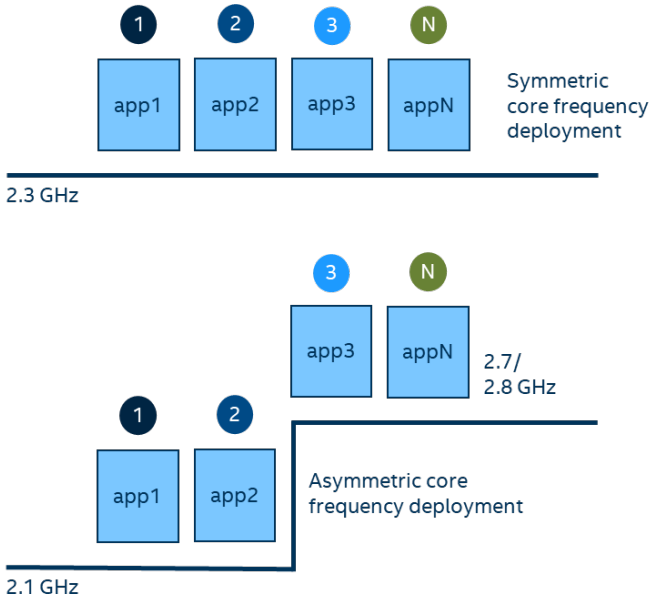

Figure 9. CPU Core Frequency Deployment Methods

Applications that consist of virtualized switches or load distribution functions in front of worker threads can benefit from the Intel SST-BF feature. With high frequencies, load distribution threads pass data extremely fast to workers.

High frequency cores are advertised as *exclusive cores* by the CPU Manager for Kubernetes. Pod manifest must request for `cmk.intel.com/exclusive-cores` to obtain one.

Intel SST-BF enabled on node is advertised by the Node Feature Discovery with the `sst_bf.enabled` label.

Example pod manifest with exclusive cores request:

```
apiVersion: v1
kind: Pod
metadata:
 name: high-priority-pod
 annotations:
 cmk.intel.com/mutate: "true"
spec:
 restartPolicy: Never
 containers:
 - name: nginx
   image: nginx:1.19.3
```

[12] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

```
  ports:
  - containerPort: 80
  resources:
    requests:
      cmk.intel.com/exclusive-cores: '1'
    limits:
      cmk.intel.com/exclusive-cores: '1'
  nodeSelector:
    feature.node.kubernetes.io/cpu-power.sst_bf.enabled: true
```

For more information about scheduling workloads on Kubernetes with Intel SST-BF support, visit https://builders.intel.com/docs/networkbuilders/intel-speed-select-technology-base-frequency-with-kubernetes-application-note.pdf

On the 3rd Generation Intel Xeon Scalable processor, it is possible to prioritize power flow to the high priority cores in the event there is a case power constraint scenario, for example, TDP ceiling hit or power supply malfunction. Power prioritization helps high frequency cores to maintain their frequency.

In BMRA deployment, Core Priority of high frequency cores is enabled by default.

For more information about Intel SST-BF technology, refer to https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html#intel-r-speed-select-technology-base-frequency-intel-r-sst-bf

4.7.2 Intel Speed Select Technology – Core Power

Intel Speed Select Technology – Core Power (Intel SST-CP), available on 3rd Generation Intel Xeon Scalable processors, allows users to define priorities in core power flow in the event there is a power constraint scenario, for example TDP ceiling hit or power supply malfunction. Power prioritization helps cores maintain their frequency, while power is drawn from cores with lower priority.

BMRA allows you to define four Intel SST-CP Classes of Service that contain:
1. Priority
2. Affected Cores
3. Minimum CPU Frequency
4. Maximum CPU Frequency

Intel SST-CP configuration can be found in `example/profile/host_vars/node1.yml`.

Intel SST-CP enabled on a node is advertised by Node Feature Discovery with `sst_cp.enabled` label.

For more information about Intel® SST-CP technology, refer to: https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html#intel-r-speed-select-technology-core-power-intel-r-sst-cp

4.7.3 Intel Speed Select Technology – Turbo Frequency (Intel SST-TF)

Intel SST-TF, available on 3rd Generation Intel Xeon Scalable processors, allows you to assign specific cores to get higher turbo frequency. This feature enables the ability to set different “All core turbo ratio limits” to cores based on the priority. By using this feature, some cores can be configured to get higher turbo frequency by designating them as high priority at the cost of lower or no turbo frequency on the low priority cores. For this reason, this feature is only useful when the system is busy utilizing all CPUs, but the user wants some configurable option to get high performance on some CPUs.

The support of Intel SST-TF depends on Intel SST-PP performance level configuration. It is possible that only a certain performance level supports Intel SST-TF. It is also possible that only the base performance level (level = 0) has the support of Intel SST-TF. Hence, first select the desired performance level to enable this feature.

For more information about Intel SST-TF technology, refer https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html#intel-r-speed-select-technology-turbo-frequency-intel-r-sst-tf.

4.7.4 Intel Speed Select Technology – Performance Profile (Intel SST-PP)

Intel SST-PP, available on 3rd Generation Intel Xeon Scalable processors, permits configuration of a server dynamically based on workload performance necessities. The working nature of Intel SST-PP is to set up CPUs that need to be online and others that need to remain offline to sustain a guaranteed frequency performance level. Intel SST-PP provides the ability to monitor changes in frequency levels dynamically.

BMRA provides three options for Intel SST-PP deployment, as described below:

- Configure Intel SST-PP with all features (Intel SST-BF, Intel SST-CP, and Intel SST-TF) in auto mode, which enables turbo frequency for all available online CPUs automatically.
- Configure Intel SST-PP with all features (Intel SST-BF, Intel SST-CP, and Intel SST-TF) with user-defined specific CPUs ranges such as "2,3,5" to prioritize those CPUs over others.

- Configure Intel SST-PP by disabling all features (Intel SST-BF, Intel SST-CP, and Intel SST-TF).

Intel SST-PP configuration can be found in `example/profile/host_vars/node1.yml`.

For more information about Intel SST-PP technology, refer to https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html#intel-r-speed-select-technology-performance-profile-intel-r-sst-pp

4.8 Security

The following sections describe some of the security features in BMRA.

4.8.1 Cluster Security

Security for the Kubernetes cluster includes the following[13]:

- Kubernetes audit trail and log backups by default
- Certs for API server access and timeouts
- Checksums for all downloaded artifacts
- RBAC enabled by default
- TLS cipher suites to help secure cluster communication
- Help secure the container registry by:
  - Limited access with TLS and basic authentication
  - Required server keys and certs signed with Kubernetes Certificate Authority (CA)
- Limited etcd permissions
- Restricted open firewall ports and subnets
- Pod Security Policies (PSP) admission controller enabled with minimal set of rules (defines a set of conditions pods must use to run within the system):
  - EventRateLimit – mitigates DoS attacks against API server
  - AlwaysPullImages – forces credential checks every time a pod image is accessed
  - NodeRestriction – limits objects that a kubelet can modify
  - PodSecurityPolicy
- Security policies on each Helm chart for deployment (collectd, NFD, TAS, and the like)

For additional security considerations, refer to:

- https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/
- https://kubernetes.io/docs/concepts/policy/pod-security-policy/
- https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#noderestriction

4.8.2 Intel® Security Libraries for Data Center (Intel® SecL – DC)

Security for cloud native applications is an increasing challenge for developers. Hardware and software suppliers in the industry are cooperatively developing the cloud architecture to deliver 5G network and edge infrastructure, which makes security an important requirement as workloads and suppliers converge.

By using Intel® SecL – DC, Hardware Root of Trust, and Secure Boot, we can create an end-to-end platform security solution for network and edge platforms. Key components of Intel® SecL – DC include the following:

- Verification service: Installed in the central control node, it gathers the secure boot signatures and maintains the platform's "trusted" or "untrusted" evaluation results. It maintains the platform trust database with established "known good" values or expected measurements. If a certain firmware or Linux kernel is found to be compromised, the policy can change the platform trust status.
- Trust agent: Installed in every physical node that needs to be monitored by the platform security, it takes the TPM ownership on the platform and reports the platform security status to the remote management module.
- Integration hub: Connects with orchestration software such as Kubernetes and contains an extension to work with K8s. It retrieves the information from the verification service and shares with the orchestration software at a specified interval.

For details on how to deploy, refer to: https://builders.intel.com/docs/networkbuilders/secure-the-network-infrastructure-secure-cloud-native-network-platforms-user-guide.pdf

4.8.3 Intel® Software Guard Extensions

Intel® Software Guard Extensions (Intel® SGX) is an Intel technology to protect select code and data from disclosure or modification. It operates by allocating hardware-protected memory where code and data reside. The protected memory area is called an *enclave*.

[13] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Data within the enclave memory can only be invoked via special instructions. This feature is only available on 3rd Generation Intel® Xeon® Scalable processors.

4.8.4 OpenSSL and QAT Engine

Network security vendors sometimes rely on the OpenSSL project, and specifically the libcrypto general purpose cryptographic library, when executing in the various cloud compute environment. To address the need to have both portability and performance, Intel offers the Intel® QAT Engine for OpenSSL as an additional option to the default library.

The Intel® QAT Engine for OpenSSL 1.1.x provides support for secure applications by directing the requested computation of cryptographic operations to the available hardware acceleration or instruction acceleration present on the platform. The engine supports both the traditional synchronous mode for compatibility with existing applications and the new asynchronous mode introduced in OpenSSL 1.1.0 to achieve maximum performance.

4.9 Security - Key Management Reference Application with Intel® SGX

Key Management Reference Application (KMRA) is a proof-of-concept software created to demonstrate the integration of Intel® Software Guard Extensions (Intel® SGX) asymmetric key capability with a hardware security model (HSM) on a centralized key server. The goal of this section is to outline BMRA setup of KMRA infrastructure with Intel SGX for private key provisioning to an Intel® SGX enclave on a 3rd Generation Intel® Xeon® Scalable processor, using the Public-Key Cryptography Standard (PKCS) #11 interface and OpenSSL. The BMRA V21.09 release highlights KMRA support on the 3rd Generation Intel® Xeon® Scalable processor only.

BMRA contains Ansible automation scripts to set up the KMRA infrastructure. This includes the setup of a centralized key server and multiple compute nodes enabled with Intel SGX. The compute nodes are provisioned with private keys into the Intel SGX enclaves to be used by workloads. This general solution can work with any application or workload. For this release, a sample NGINX workload/application Ansible script is provided as an example of how the KMRA infrastructure can be used. The NGINX workload uses the private keys from the Intel SGX enclave to establish TLS connections. The private key is never in the clear and the certificate signing happens more securely inside the enclave.

Step-by-step instructions for deploying KMRA with BMRA are detailed in Workloads and Application Examples. The instructions also provide information on how to deploy the NGINX workload and configure it to use the private key more securely from inside the enclave to establish TLS connections.

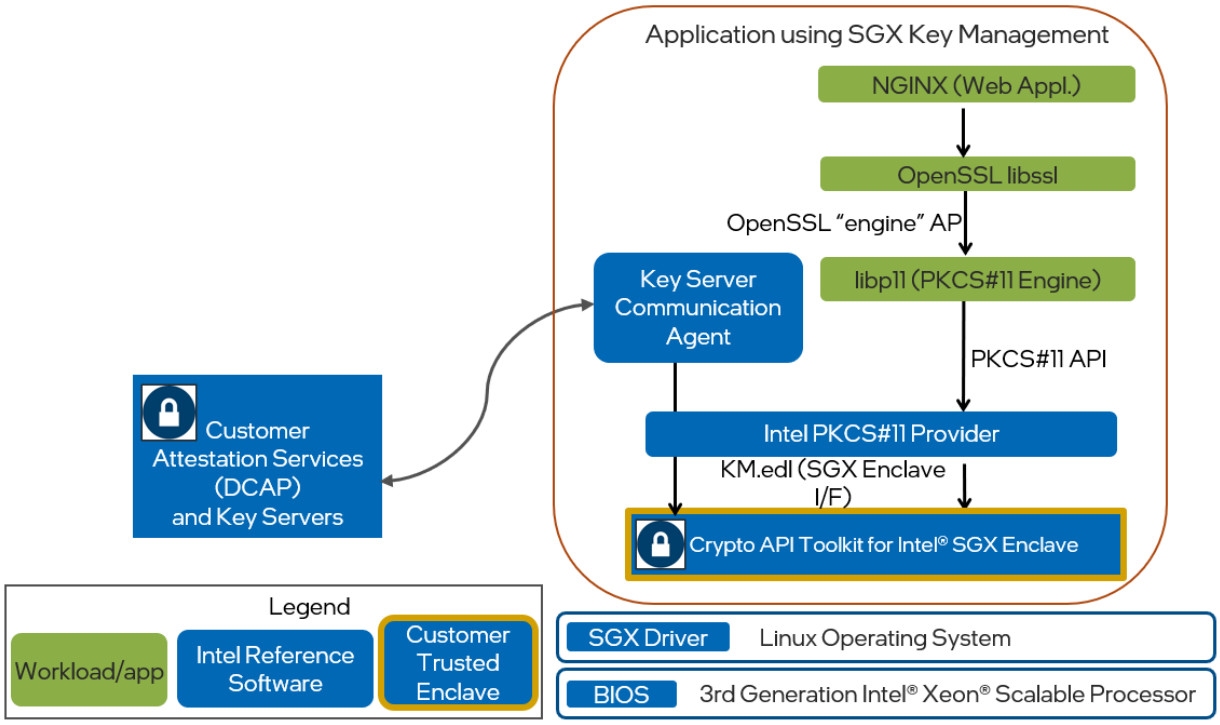

Figure 10. Key Management Reference Application Infrastructure with Intel® SGX

For more information, see Intel® Software Guard Extensions (Intel® SGX) – Key Management on the 3rd Generation Intel® Xeon® Scalable Processor Technology Guide and Intel® Software Guard Extensions (Intel® SGX) – NGINX Private Key on 3rd Generation Intel® Xeon® Scalable Processor User Guide.

4.10 Intel® Server GPU

The Intel® Server GPU is the first discrete graphics processing unit for data centers based on the new Intel Xe architecture with support for MPEG-2, AVC, HEVC, and VP9 transcoding plus AV1 decode. The Intel® Server GPU is a high-density media processing accelerator that enables users, via configurable presets, to choose the video quality/density setting that meets their requirements. The Intel® Server GPU programming is supported via open-source software including drivers, APIs, and developer tools such as the Intel® Media SDK and FFmpeg plugin. The Intel® Server GPU supports low level programmability including frame and subframe level controls.

The Intel® Server GPU is based on a low-power discrete system-on-chip (SoC) design, with a 128-bit wide pipeline and 8 GB of dedicated onboard low-power DDR4 memory. Four GPU SoCs are packaged together in a three-quarter length, full height x16 PCIe Gen3 add-in card from H3C, with a target configuration of up to four cards per server.

The Open Visual Cloud is a set of open-source software stacks (with full end-to-end sample pipelines) for media, analytics, graphics, and immersive media, optimized for cloud native deployment on commercial-off-the-shelf x86 CPU architecture. An Open Visual Cloud reference application for OTT transcoding and delivery via CDN, on both Intel® Xeon® and Intel® Server GPU, is available in the Open Visual Cloud GitHub site https://github.com/OpenVisualCloud/CDN-Transcode-Samplev.

For more details on supported operating systems and required firmware for the Intel® Server GPU, work with your local Intel sales representative. For more details on the product, see the following links:
https://www.intel.com/content/www/us/en/products/discrete-gpus/server-graphics-card.html

https://www.intel.com/content/www/us/en/benchmarks/server/graphics/intelservergpu.html

https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/server-graphics-card-product-brief.html

https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/server-graphics-solution-brief.html

https://ark.intel.com/content/www/us/en/ark/products/210576/intel-server-gpu.html

4.11 Observability

The BMRA Telemetry software stack consists of telemetry collectors deployed on every node, a telemetry time-series database that pulls the metrics from collectors, telemetry visualization software, and a policy agent that influences cluster scheduling decisions.

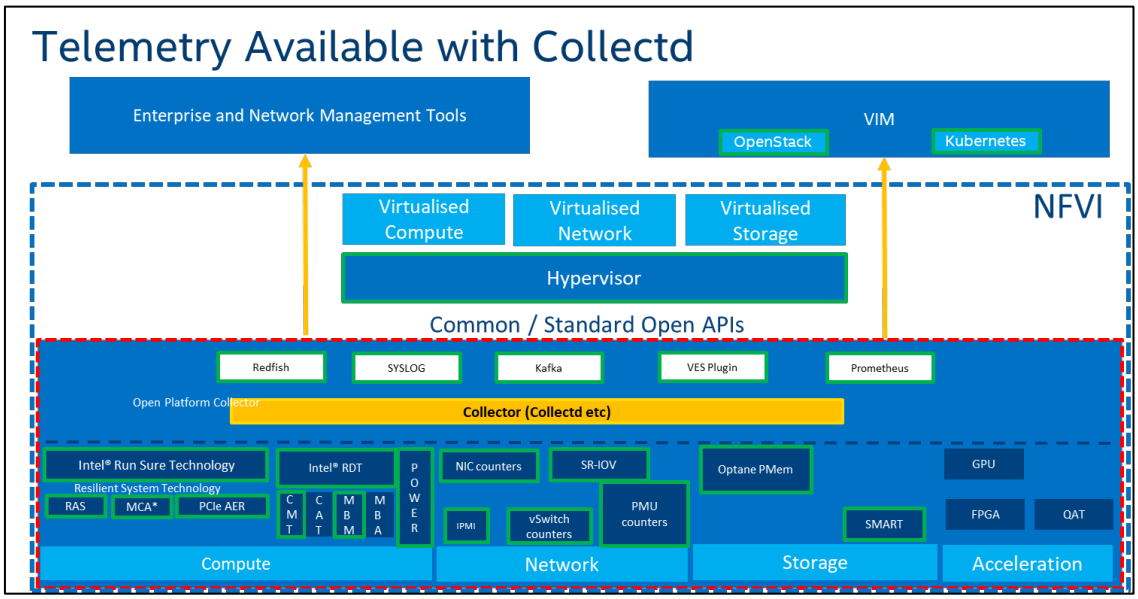

Figure 11. Platform Telemetry Available with Collectd

4.11.1 Observability Components Overview

This section describes the components shown in the previous diagram, including Telegraf, Collectd, Node Exporter, Prometheus, Grafana, and Prometheus adapter.

4.11.1.1 Telegraf

Telegraf is an open-source plugin-based server agent that you can use to collect custom application metrics, logs, network performance data, platform metrics, and more. Developed by InfluxData, Telegraf's pluggable architecture uses input plugins for

collecting metrics and output plugins for sending metrics to various endpoints, such as time series databases like Prometheus and Influx DB or data streaming services like Kafka.

For more information on Telegraf and its available plugins, visit https://www.influxdata.com/time-series-platform/telegraf/.

4.11.1.2 collectd

collectd is a UNIX daemon responsible for collecting a wide spectrum of platform telemetry. It has a modular architecture and data acquisition depends on loaded plugins. Plugins that are loaded with collectd depend on an installed profile.

Table 5. Collectd Plugins

PLUGIN	PROFILE
cpu	All
cpufreq	All
disk	All
ipmi	All
numa	All
smart.conf	All
ethstat	All
netlink	All
intel_pmu	All
intel_rdt	All
pkgpower.py	All
dpdkevents	Full
dpdkstat	Full
hugepages	Full
ovs_events	Full
ovs_pmd_stats	Full
ovs_stats	Full

For detailed descriptions of the plugins and metrics available, visit https://collectd.org/ and https://wiki.opnfv.org/display/fastpath/Barometer+Home.

For more information about pkgpower.py plugin, visit https://github.com/intel/CommsPowerManagement/blob/master/power.md

Note: To enable intel_pmu plugin, set the `enable_intel_pmu_plugin: true` configuration variable.

Note: When intel_rdt and intel_pmu plugins run on a node at the same time, the following metrics are measured incorrectly:

- Instructions
- Cycles
- Instructions per cycle (IPC)
- Cycles per instruction (CPI)

These metrics are measured incorrectly because both plugins count the same metric using different means (MSR write vs. perf). You can choose to disable one plugin or the other by commenting the lines with – `intel_pmu.conf` or – `rdt.conf` in the `roles/collectd-install/default/main.yml` file.

4.11.1.3 Node Exporter

Node Exporter is a Prometheus exporter for hardware and OS metrics exposed by NIX kernels, written in Go with pluggable metric collectors. Node Exporter is run with its default configuration on every node.

4.11.1.4 Prometheus

Prometheus is an open-source systems-monitoring server that scrapes and stores time series data.

4.11.1.5 Grafana

Grafana is an open-source telemetry visualization tool, which is available under the HTTPS endpoint: `https://localhost:30000` on any of the cluster nodes. Due to security reasons, this port is not exposed outside the cluster by default.

Note: Grafana is deployed with default credentials of admin/admin. After first login, the user is asked to change the password. We recommend that you change this as soon as possible.

4.11.1.6 Prometheus Adapter

Prometheus Adapter is an implementation of the Kubernetes resource metrics API and custom metrics API. It provides Prometheus metrics to the Telemetry Aware Scheduler.

4.11.2 Platform Telemetry Security

Platform Telemetry is an asset that must be protected. This section provides an overview of telemetry security.

4.11.2.1 Data at Rest Security

Note: BMRA does not provide any data at rest security for telemetry. We recommend that you provide proper means of data at rest security, such as self-encrypting drives.

4.11.2.2 Data in Transit Security

BMRA helps secure the telemetry traffic between nodes using TLS. Prometheus verifies a collector's certificate on connection using an in-cluster CA certificate while the collector helps ensure that the service account token provided with HTTP connection allows it to read a "/metrics" endpoint. Service Account Token verification is provided by the Kubernetes API Server.

5 Reference Architecture Hardware Components and BIOS

For all BMRA Configuration Profiles, this section provides a menu of all possible hardware components for control node and worker node as well as the BIOS components available.

5.1 Hardware Component List for Control Node

This table lists the hardware options for control nodes, which are responsible for managing the worker nodes in the cluster.

Table 6. Hardware Options for Control Node – 2nd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
2nd Generation Intel® Xeon® Scalable processors	Intel® Xeon® Gold 5218 or 5218N processor at 2.3 GHz, 16 C/32 T, 125 W, or higher number Intel® Xeon® Gold or Platinum CPU SKU	Required
Memory	DRAM only configuration: 192 GB (12 x 16 GB DDR4 2666 MHz)	Required
Network Adapter	Option 1: Dual Port 25 GbE Intel® Ethernet Network Adapter XXV710-DA2 SFP28+, or	Required
	Option 2: Dual Port 10 GbE Intel® Ethernet Converged Network Adapter X710-DA2 SFP+, or	
	Option 3: Dual Port 10 GbE Intel® Ethernet Converged Network Adapter X520-DA2 SFP+	
Intel® QAT	Intel® QuickAssist Adapter 8970 (PCIe) AIC or equivalent third-party Intel® C620 Series Chipset Intel® QAT enabled PCIe AIC, with minimum 8 lanes of PCIe connectivity	Recommended
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® NVMe SSD DC P4510 Series at 2 TB or equivalent (Recommended NUMA Aligned)	Recommended
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	N/A	

Table 7. Hardware Options for Control Node – 3rd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
3rd Generation Intel Xeon Scalable processors	Intel® Xeon® Gold 5318N processor at 2.1 GHz, 20 C/40 T, 135W or higher number Intel® Xeon® Gold or Platinum CPU SKU	Required

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
Memory	256 GB DRAM (16x 16 GB DDR4, 2666 MHz)	Required
Network Adapter	Dual Port 100 GbE Intel® Ethernet Network Adapter E810-CQDA2 QSFP28	Required
Intel® QAT	Intel® QuickAssist Adapter 8960 or 8970 (PCIe*) AIC or equivalent third-party Intel® C620 Series Chipset	Recommended
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® SSD D7-P5510 Series at 3.84 TB or equivalent drive (recommended NUMA aligned)	Recommended
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	N/A	

5.2 Hardware Component List for Worker Node Base

A Kubernetes cluster typically consists of multiple worker nodes managed by one or more Kubernetes control nodes.

This table lists the hardware options for worker nodes in the “base” configuration. If your configuration needs improved processing, you may choose to use the “plus” configuration instead. See the next section for details.

Table 8. Hardware Components for Worker Node Base – 2nd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
2nd Generation Intel® Xeon® Scalable processors	Intel® Xeon® Gold 6230 processor @ 2.1 GHz or 6230N CPU @ 2.3 GHz 20 C/40 T, 125 W, or higher number Intel® Xeon® Gold or Platinum CPU SKU	Required
Memory	Option 1: DRAM only configuration: 384 GB (12 x 32 GB DDR4 2666 MHz)	Required
	Option 2: DRAM only configuration: 384 GB (24 x 16 GB DDR4 2666 MHz)	
	Option 3: DRAM + Intel® Optane™ Persistent Memory DRAM: 192 GB (12x 16 GB DDR4, 2666 MHz)	
Intel® Optane™ Persistent Memory	512 GB (4x 128 GB Intel® Optane™ persistent memory in 2-1-1 Topology)	Recommended
Network Adapter	Option 1: Dual Port 100 GbE Intel® Ethernet Network Adapter E810-CQDA2 QSFP28	Required
	Option 2: Intel® Ethernet Network Adapter XXV710-DA2 QSFP28	
Intel® QAT	Intel® QuickAssist Adapter 8970 (PCIe) AIC or equivalent third-party Intel® C620 Series Chipset Intel® QAT enabled with minimum 8 lanes of PCIe connectivity	Required
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® NVMe SSD DC P4510 Series P4510 at 2 TB or equivalent (Recommended NUMA Aligned)	Required
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	N/A	

Table 9. Hardware Components for Worker Node Base – 3rd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
3rd Generation Intel Xeon Scalable processors	Intel® Xeon® Gold 5318N processor at 2.1 GHz, 24 C/48 T, 150 W, or higher number Intel® Xeon® Gold or Platinum CPU SKU	Required
Memory	Option 1: DRAM only configuration: 256 GB (8 x 32 GB DDR4, 2666 MHz)	Required
	Option 2: DRAM only configuration: 256 GB (16 x 16 GB DDR4, 2666 MHz)	
Intel® Optane™ Persistent Memory	512 GB (4x 128 GB Intel® Optane™ persistent memory in 2-1-1 Topology)	Recommended

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
Network Adapter	Option 1: Intel® Ethernet Network Adapter E810-CQDA2	Required
	Option 2: Intel® Ethernet Network Adapter E810-XXVDA-2	
Intel® QAT	Intel® QuickAssist Adapter 8960 or 8970 (PCIe*) AIC or equivalent third-party Intel® C620 Series Chipset	Required
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® SSD D7-P5510 Series at 3.84 TB or equivalent drive (recommended NUMA aligned)	Required
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	N/A	

5.3 Hardware Component List for Worker Node Plus

A Kubernetes cluster typically consists of multiple worker nodes managed by one or more Kubernetes control nodes.

This table lists the hardware options for worker nodes in the “plus” configuration, which helps improves the processing capability due to more powerful CPU, more memory, more disk space, and an amazingly fast network.

Table 10. Hardware Components for Worker Node Plus – 2nd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
2nd Generation Intel® Xeon® Scalable processors	Intel® Xeon® Gold 6252 processor @ 2.1 GHz or 6252N processor @ 2.3 GHz 24 C/48 T, 150 W, or higher number Intel® Xeon® Gold/Platinum CPU SKU	Required
Memory	Option 1: DRAM only configuration: 384 GB (12 x 32 GB DDR4 2666 MHz)	Required
	Option 2: DRAM only configuration: 384 GB (24 x 16 GB DDR4 2666 MHz)	
	Option 3: DRAM + Intel® Optane™ persistent memory DRAM: 192 GB (12 x 16 GB DDR4 2666 MHz)	
Intel® QAT	Intel® C620 Series Chipset integrated on base board Intel® C627/C628 Chipset, integrated with NUMA connectivity to each CPU or minimum 16 Peripheral Component Interconnect express (PCIe) lane connectivity to one CPU	Required
Intel® Optane™ Persistent Memory	Option 1: 1 TB (8x 128 GB Intel® Optane™ persistent memory in 2-2-1 Topology)	Recommended
	Option 2: 1.5 TB (12x 128 GB Intel® Optane™ persistent memory in 2-2-2 Topology)	
Network Adapter	Option 1: Dual Port 25 GbE Intel® Ethernet Network Adapter X710-DA4 SFP+, or	Required
	Option 2: Dual Port 100 GbE Intel® Ethernet Network Adapter E810-CQDA2 QSFP28	
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® NVMe SSD DC P4510 Series at 2 TB or equivalent (Recommended NUMA Aligned)	Required
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	N/A	

Table 11. Hardware Components for Worker Node Plus – 3rd Generation Intel Xeon Scalable Processor

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
3rd Generation Intel Xeon Scalable processors	Intel® Xeon® Gold 6338N CPU @ 2.2 GHz 32 C/64 T, 185 W, or higher number Intel® Xeon® Gold or Platinum CPU SKU	Required
Memory	Option 1: DRAM only configuration: 512 GB (16x 32 GB DDR4, 2666 MHz)	Required
	Option 2: DRAM only configuration: 512 GB (32x 16 GB DDR4, 2666 MHz)	
Intel® QAT	Intel® C620 Series Chipset integrated on base board Intel® C627/C628 Chipset, integrated with NUMA connectivity to each CPU or minimum 16 Peripheral Component Interconnect express (PCIe) lane connectivity to one CPU	Required

INGREDIENT	REQUIREMENT	REQUIRED/ RECOMMENDED
Intel® Optane™ Persistent Memory	Option 1: 1 TB (8x 128 GB Intel® Optane™ persistent memory in 8+4 Topology)	Recommended
	Option 2: 2 TB (16x 128 GB Intel® Optane™ persistent memory in 8+8 Topology)	
Network Adapter	Option 1: Intel® Ethernet Network Adapter E810-CQDA2	Required
	Option 2: Intel® Ethernet Network Adapter E810-2CQDA2	
Storage (Boot Drive)	Intel® SATA Solid State Drive D3 S4510 at 480 GB or equivalent boot drive	Required
Storage (Capacity)	Intel® SSD D7-P5510 Series at 4 TB or equivalent drive (recommended NUMA aligned)	Recommended
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Preboot Execution Environment (PXE) and Operation, Administration, and Management (OAM)	Required
	1/10 Gbps port for Management Network Adapter	Required
Additional Plug-in cards	Intel® Server Graphics 1 card	Optional

5.4 Hardware BOMs for all Configuration Profiles

The following tables list the hardware BOMs for Control Nodes, Worker Node Base, and Worker Node Plus.

Choose your controller profile from the three available profiles (Controller_xGen_1, Controller_xGen_2, or Controller_xGen_3) based on your BIOS Profile (Energy Balance, Deterministic, or Max Performance, respectively).

The profiles for Worker Nodes vary with respect to Network Interface card, Intel® QuickAssist Technology, and BIOS profiles. You may choose based on the requirement for the workloads to be run on the worker nodes.

Table 12. Control Node Hardware Setup for all Configuration Profiles – 2nd Generation Intel Xeon Scalable Processor

NAME	Controller_2ndGen_1	Controller_2ndGen_2	Controller_2ndGen_3
Platform	S2600WFQ	S2600WFQ	S2600WFQ
CPU/node	2x 5218 or 2x 5218N	2x 5218 or 2x 5218N	2x 5218 or 2x 5218N
Mem	192 GB	192 GB	192 GB
Intel Optane Persistent Memory	Recommended	Recommended	Recommended
Network Adapter	2x XXV710-DA2 or 2x X710-DA2 or 2x X520-DA2	2x XXV710-DA2	2x XXV710-DA2
Storage (Boot Media)	Required - 2x	Required - 2x	Required - 2x
Storage (Capacity)	Recommended - 2x (1 per NUMA)	Recommended - 2x (1 per NUMA)	Recommended - 2x (1 per NUMA)
LOM	No	No	No
Intel® QAT	Recommended	N/A	N/A
BIOS Configuration			
Intel® HT Technology enabled	Yes	Yes	Yes
Intel® VT-x enabled	No	Yes	Yes
Intel® VT-d enabled	No	Yes	Yes
BIOS Profile	Energy Balance	Deterministic	Max Performance
Virtualization enabled	No	Yes	Yes

Table 13. Control Node Hardware Setup for all Configuration Profiles – 3rd Generation Intel Xeon Scalable Processor

NAME	Controller_3rdGen_1	Controller_3rdGen_2	Controller_3rdGen_3
Platform	M50CYP	M50CYP	M50CYP
CPU/node	2x 5318N 20c	2x 5318N 20c	2x 5318N 20c
Mem	256 GB	256 GB	256 GB
Intel Optane Persistent Memory	Recommended	Recommended	Recommended
Network Adapter	2x E810-CQDA2	2x E810-CQDA2	2x E810-CQDA2
Storage (Boot Media)	Required - 2x	Required - 2x	Required - 2x
Storage (Capacity)	Recommended - 2x (1 per NUMA)	Recommended - 2x (1 per NUMA)	Recommended - 2x (1 per NUMA)
LOM	No	No	No
Intel® QAT	Recommended	N/A	N/A
BIOS Configuration			
Intel® HT Technology enabled	Yes	Yes	Yes
Intel® VT-x enabled	No	Yes	Yes
Intel® VT-d enabled	No	Yes	Yes
BIOS Profile	Energy Balance	Deterministic	Max Performance
Virtualization enabled	No	Yes	Yes

Table 14. Worker Node Base Hardware Setup for all Configuration Profiles – 2nd Generation Intel Xeon Scalable Processor

NAME	Worker_2ndGen_Base_1	Worker_2ndGen_Base_2	Worker_2ndGen_Base_3
Platform	S2600WFQ	S2600WFQ	S2600WFQ
CPU/node	2x 6230 or 6230N	2x 6230 or 6230N	2x 6230 or 6230N
Mem	384 GB	384 GB	384 GB
Intel Optane Persistent Memory	Recommended – 512 GB	Recommended – 512 GB	Recommended – 512 GB
Network Adapter	2x XXV710-DA2 or 2x E810-CQDA2	2x XXV710-DA2	2x XXV710-DA2
Storage (Boot Media)	Required - 2x	Required - 2x	Required - 2x
Storage (Capacity)	Required- 2x (1 per NUMA)	Required- 2x (1 per NUMA)	Required- 2x (1 per NUMA)
LOM	No	No	Yes
Intel® QAT	No	Optional	Yes
Additional Plug-in cards	No	No	No
BIOS Configuration			
Intel® HT Technology enabled	Yes	Yes	Yes
Intel® VT-x enabled	Yes	Yes	Yes
Intel® VT-d enabled	Yes	Yes	Yes
BIOS Profile	Energy Balance	Deterministic	Max Performance
Virtualization enabled	No	Yes	Yes

Table 15. Worker Node Plus Hardware Setup for all Configuration Profiles – 2nd Generation Intel Xeon Scalable Processor

NAME	Worker_2ndGen_Plus_1	Worker_2ndGen_Plus_2
Platform	S2600WFQ	S2600WFQ
CPU/node	2x 6252 or 6252N	2x 6252 or 6252N
Mem	384 GB	384 GB
Intel Optane Persistent Memory	Recommended – 1 TB/1.5 TB	Recommended – 1 TB/1.5 TB
Network Adapter	2x E810-CQDA2	2x E810-CQDA2
Storage (Boot Media)	Required – 256 GB	Required – 256 GB
LOM	No	Yes
Intel® QAT	No	Yes
Additional Plug-in cards	No	No
BIOS Configuration		
Intel® HT Technology enabled	Yes	Yes
Intel® VT-x enabled	Yes	Yes
Intel® VT-d enabled	Yes	Yes
BIOS Profile	Deterministic	Max Performance
Virtualization enabled	Yes	Yes

Table 16. Worker Node Base Hardware Setup for all Configuration Profiles – 3rd Generation Intel Xeon Scalable Processor

NAME	Worker_3rdGen_Base_1	Worker_3rdGen_Base_2	Worker_3rdGen_Base_3
Platform	M50CYP	M50CYP	M50CYP
CPU/node	2x 5318N 24c	2x 5318N 24c	2x 5318N 24c
Mem	512 GB	512 GB	512 GB
Intel Optane Persistent Memory	Recommended – 512 GB	Recommended – 512 GB	Recommended – 512 GB
Network Adapter	2x E810-CQDA2	2x E810-CQDA2	2x E810-2CQDA2 or 4x E810-CQDA2
Storage (Boot Media)	Required - 2x	Required - 2x	Required - 2x
Storage (Capacity)	Required- 2x (1 per NUMA)	Required- 2x (1 per NUMA)	Required- 2x (1 per NUMA)
LOM	No	Yes	No
Intel® QAT	No	Yes	Optional
Additional Plug-in cards	No	No	No
BIOS Configuration			
Intel® HT Technology enabled	Yes	Yes	Yes
Intel® VT-x enabled	Yes	Yes	Yes
Intel® VT-d enabled	Yes	Yes	Yes
BIOS Profile	Energy Balance	Max Performance	Deterministic
Virtualization enabled	No	Yes	Yes

Table 17. Worker Node Plus Hardware Setup for all Configuration Profiles – 3rd Generation Intel Xeon Scalable Processor

NAME	Worker_3rdGen_Plus_1	Worker_3rdGen_Plus_2	Worker_3rdGen_Plus_3
Platform	M50CYP	M50CYP	M50CYP
CPU/node	2x 6338N 32c	2x 6338N 32c	2x 6338N 32c

NAME	Worker_3rdGen_Plus_1	Worker_3rdGen_Plus_2	Worker_3rdGen_Plus_3
Mem	512 GB	512 GB	512 GB
Intel Optane Persistent Memory	Recommended – 512 GB	Recommended – 512 GB	Recommended – 512 GB
Network Adapter	2x E810-2CQDA2 or 4x E810-CQDA2	2x E810-2CQDA2	2x E810-2CQDA2 or 4x E810-CQDA2
Storage (Boot Media)	Required - 2x	Required - 2x	Required - 2x
Storage (Capacity)	Required- 4x (2 per NUMA)	Required- 4x (2 per NUMA)	Required- 4x (2 per NUMA)
LOM	Yes	Yes	No
Intel® QAT	Yes	No	Optional
Additional Plug-in cards	No	Intel Server GPU	No
BIOS Configuration			
Intel® HT Technology enabled	Yes	Yes	Yes
Intel® VT-x enabled	Yes	Yes	Yes
Intel® VT-d enabled	Yes	Yes	Yes
BIOS Profile	Max Performance	Max Performance	Deterministic
Virtualization enabled	Yes	Yes	Yes

5.5 Platform BIOS

This section provides BIOS Configuration Profiles for each of the BMRA Configuration Profiles. For details on how the BIOS configuration should be set per each Configuration Profile, go to tables in Section 5.4.

For more information about BIOS settings, visit https://www.intel.com/content/dam/support/us/en/documents/server-products/Intel_Xeon_Processor_Scalable_Family_BIOS_User_Guide.pdf.

Table 18. Platform BIOS Settings for 2nd Generation Intel® Xeon® Scalable Processor

MENU (ADVANCED)	PATH TO BIOS SETTING	BIOS SETTINGS	ENERGY BALANCE	MAX PERFORMANCE	DETERMINISTIC
Advanced	Processor Configuration	Intel® Hyper-Threading Tech	Enabled	Enabled	Enabled
		Intel® Virtualization Technology	Enabled	Enabled	Enabled
	Integrated IO Configuration	Intel® VT for Directed I/O	Enabled	Enabled	Enabled
Advanced/Power Configuration	Power and Performance	CPU Power and Performance Policy	Balanced Performance	Performance	Performance
		Workload Configuration	I/O sensitive	I/O sensitive	I/O sensitive
	CPU P-state control	Enhanced Intel SpeedStep® Technology	Enabled	Enabled	Disabled* [Read footnote]
		Activate PBF	Disabled	Enabled	Enabled
		Configure PBF	Disabled	Disabled	Disabled
		Intel® Turbo Boost Technology	Enabled	Enabled	Disabled* [Read footnote]
		Energy Efficient Turbo	Enabled	Disabled	N/A
		Intel Configurable TDP	Disabled	Disabled	Disabled
	Hardware P-states	Hardware P-states	Native Mode with no legacy Support	Disabled** [Read footnote]	Disabled** [Read footnote]
		EPP Enable	Enabled	Enabled	Enabled
		RAPL Prioritization	Disabled	Enabled	Enabled
	CPU C-state Control	Package C-state	C6 Retention	C6 Retention	C0/C1 State
		C1E	Enabled	Enabled	Disabled
		Processor C6	Enabled	Enabled	Disabled
	Uncore Power Management	Uncore Frequency scaling	Enabled	Disabled	Disabled
		Performance P-limit	Enabled	Disabled	Disabled
Advanced	Memory Configuration	IMC Interleaving	2-way interleave	2-way Interleave	2-way Interleave
	System Acoustic and Performance Configuration	Set Fan Profile	Acoustic	Performance	Performance
GPU	GPU Fz	Lock 900Mhz	Optional	Optional	Optional

* Enabled in the case where Intel® SST-BF is enabled to allow for configuration of individual core speeds.

** "Native Mode with No Legacy Support" where Intel® SST-BF need to be enabled

Table 19. Platform BIOS Settings for 3rd Generation Intel® Xeon® Scalable Processor

MENU (Advanced)	Path to BIOS Setting	BIOS Setting	Energy Balance	Max Performance with Turbo	Deterministic
Socket Configuration	Processor Configuration	Hyper-Threading	Enable	Enable	Enable
		XAPIC	Enable	Enable	Enable
		VMX	Enable	Enable	Enable
		Uncore frequency scaling	Enable	Enable	Disable
		Uncore frequency	800-2400	800-2400	2400
Power Configuration	Power and Performance	CPU Power and Performance Policy	Balance Performance	Performance	Performance
		Workload Configuration	I/O sensitive	I/O sensitive	I/O sensitive
	CPU P State Control	EIST PSD Function	HW_ALL	HW_ALL	HW_ALL
		Boot Performance Mode	Max. Performance	Max. Performance	Max. Performance
		AVX License Pre-Grant	Disable	Disable	Disable
		AVX ICCP Pre Grant Level	NA	NA	NA
		AVX P1	Nominal	Nominal	Nominal
		Energy Efficient Turbo	Enable	Enable	Disable
		WFR Uncore GV rate Reduction	Enable	Enable	Enable
		GPSS timer	500us	0us	0us
		Intel Turbo Boost Technology	Enable	Enable	Disable
		Intel SpeedStep® Technology (P-states)	Enable	Enable	Disable
	Frequency Prioritization	RAPL Prioritization	Enable	Disable	Disable
	Hardware PM State Control	Hardware P-States	Native Mode with no legacy Support	Disable	Disable
		EPP enable	Enable	Disable	Disable
	CPU C State Control	Enable Monitor Mwait	Enable	Enable	Enable
		CPU C1 Auto Demotion	Enable	Disable	Disable
		CPU C1 Auto unDemotion	Enable	Disable	Disable
		CPU C6 Report	Enable	Enable	Disable
		Processor C6	Enable	Enable	Disable
		Enhanced Halt State (C1E)	Enable	Enable	Disable
		OS ACPI Cx	ACPI C2	ACPI C2	ACPI C2
	Energy Performance Bias	Power Performance Tuning	OS Controls EPB	OS Controls EPB	OS Controls EPB

		ENERGY_PERF_BIAS_CFG mode	Performance	Performance	Performance
		Workload Configuration	I/O Sensitive	I/O Sensitive	I/O Sensitive
	Package C State Control	Package C State	C6 Retention	C6 Retention	C0/C1 State
		Dynamic L1	Enable	Disable	Disable
		Package C-state Latency Negotiation	Disable	Disable	Disable
		PKGC_SA_PS_CRITERIA	Disable	Disable	Disable
Memory Configuration		Memory Configuration	2-way interleave	2-way interleave	2-way interleave
		Enforce POR	Enable	Enable	Enable
Platform Configuration	Miscellaneous Configuration	Serial Debug Message Level	Minimum	Minimum	Minimum
	PCI Express* Configuration	PCIe* ASPM Support	Per Port	Per Port	Per Port
	PCI Express* Configuration	PCIe* ASPM	Enable	Disable	Disable
	PCI Express* Configuration	ECRC generation and checking	Enable	Enable	Enable
Server Management		Resume on AC Power Loss	Power On	Power On	Power On
System Acoustic and Performance Configuration		Set Fan Profile	Acoustic	Performance	Performance

Use the following table to configure the BIOS settings to use Intel SST-BF, Intel SST-TF, and Intel SST-PP in 3rd Generation Intel Xeon Scalable Processor systems.

Table 20. BIOS Settings to Enable Intel SST-BF, Intel SST-TF, and Intel SST-PP

BIOS SETTING	STATUS
Hardware PM State Control	
Scalability	Disable
Hardware PM Interrupt	Disable
CPU P-state	
Dynamic SST-PP	Enable
Speed Step (P states)	Enable
Activate SST-BF	Enable
Configure SST-BF	Enable
EIST PSD Function	HW_All
Turbo	Enable
Energy Efficient Turbo	Enable
Boot Performance	Max
Freq: Prioritization AC	
SST-CP	Enable

In BIOS, the configuration paths might be slightly different, depending on platform, but the key settings are as follows and must be performed in order.

Table 21. BIOS Settings to Enable Intel SGX on 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

BIOS SETTING	STATUS
Socket Configuration > Processor Configuration > Total Memory Encryption (TME)	Enable

BIOS SETTING	STATUS
Socket Configuration > Common RefCode Configuration > UMA-Based Clustering	Disable (All2All)
Socket Configuration > Processor Configuration > SW Guard Extensions (SGX)	Enable

6 Reference Architecture Software Components

This section describes the software version details.

Table 22. Software Components

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
OS	CentOS 8.3 Kernel version: 4.18.0-240.el8.x86_64	https://www.centos.org/
	Ubuntu 20.04 Kernel version: 5.4.0-26-generic (20.04.2)	https://www.ubuntu.com
	Ubuntu 21.04 Kernel version: 5.11.0-16-generic	
	RHEL 8.3 Kernel version: 4.18.0-240.el8.x86_64	https://www.redhat.com/
	RHEL 8.4 Kernel version: 4.18.0-305.el8.x86_64	
Data Plane Development Kit	DPDK 21.08	https://core.dpdk.org/download/
Open vSwitch with DPDK	OVS-DPDK v2.16.0	https://github.com/openvswitch/ovs
Vector Packet Processing	VPP 19.04	https://docs.fd.io/vpp/
Telegraf	Latest	https://github.com/intel/observability-telegraf
CollectD	opnfv/barometer-collectd: latest	https://www.collectd.org/
Grafana	8.1.2	https://www.grafana.com/
Prometheus	2.29.1	
Ansible	Ansible v2.9.20	https://www.ansible.com/
BMRA Ansible Playbook	v21.09	https://github.com/intel/container-experience-kits
Python	Python 3.6.x for RHEL 8/CentOS 8 Python 3.8.x for Ubuntu 20.04 and Python 3.9.x for Ubuntu 21.04	https://www.python.org/
Kubespray	2.16	https://github.com/kubernetes-sigs/kubespray
Docker	Docker 19.03	https://www.docker.com/
Containerd	Containerd 1.4.6	
CRI-O	CRI-O 1.21.3	
Container orchestration engine	Kubernetes v1.21.x	https://github.com/kubernetes/kubernetes
	Kubernetes v1.20.x	
	Kubernetes v1.19.x	
CPU Manager (native to Kubernetes)	Available natively in K8s	N/A

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
CPU Manager for Kubernetes	CPU Manager for Kubernetes v1.5.1	https://github.com/intel/CPU-Manager-for-Kubernetes Container image requires GNU libc 2.29 or newer to run CPU Manager for Kubernetes.
Telemetry Aware Scheduling	TAS 0.4	https://github.com/intel/telemetry-aware-scheduling
k8s-prometheus-adapter	0.8.4	
k8s node-exporter	1.2.2	
k8s prometheus-operator	0.50.0	https://github.com/prometheus-operator/kube-prometheus
k8s kube-rbac-proxy	0.11.0	
Node Feature Discovery	NFD 0.9.0	https://github.com/kubernetes-sigs/node-feature-discovery
Multus CNI	Multus CNI v3.7	https://github.com/intel/multus-cni
SR-IOV CNI	SR-IOV CNI v2.6.1	https://github.com/intel/sriov-cni
SR-IOV network device plugin	SR-IOV network device plugin v3.3.2	https://github.com/intel/sriov-network-device-plugin
SR-IOV Network Operator	master	https://github.com/openshift/sriov-network-operator
Device Plugins Operator	v0.21.0	https://github.com/intel/intel-device-plugins-for-kubernetes
Istio Operator	1.11.1	https://github.com/istio/istio/releases/download/
QAT device plugin	v0.21.0	https://github.com/intel/intel-device-plugins-for-kubernetes
GPU device plugin	v0.21.0	https://github.com/intel/intel-device-plugins-for-kubernetes
Intel® SGX device plugin	v0.21.0	https://github.com/intel/intel-device-plugins-for-kubernetes
Userspace CNI	Userspace CNI v1.3	https://github.com/intel/userspace-cni-network-plugin
Bond CNI plugin	Bond CNI plugin v1.0	https://github.com/intel/bond-cni
Intel® SecL – DC	v1.6	https://01.org/intel-secl
Intel® Ethernet Drivers	i40e v2.16.11 ice v1.6.4 iavf v4.2.7	https://sourceforge.net/projects/e1000/files/i40e%20stable/2.16.11/ https://sourceforge.net/projects/e1000/files/ice%20stable/1.6.4/ https://sourceforge.net/projects/e1000/files/iavf%20stable/4.2.7/
Intel® Ethernet NVM Update Package 700 Series	v8.4	https://www.intel.com/content/www/us/en/download/18190/non-volatile-memory-nvm-update-utility-for-intel-ethernet-network-adapter-700-series.html
Intel® Ethernet NVM Update Package 800 Series	v3.00	https://downloadmirror.intel.com/29738/eng/e810_nvmupdatepackage_v3_00_linux.tar_.gz
DDP Profiles	Dynamic Device Personalization for Intel® Ethernet 700 Series Version 25.4	https://downloadmirror.intel.com/27587/eng/gtp.zip https://downloadmirror.intel.com/28940/eng/mplsogreudp.zip https://downloadmirror.intel.com/28040/eng/ppp-oe-ol2tpv2.zip https://downloadmirror.intel.com/29446/eng/esp-ah.zip https://downloadmirror.intel.com/29780/eng/ecpri.zip
	1.3.30.0	https://downloadcenter.intel.com/download/29889/Intel-Ethernet-800-Series-Telecommunication-Comms-Dynamic-Device-Personalization-DDP-Package
Intel® QAT Drivers	1.7.L.4.14.0-00031	https://downloadmirror.intel.com/30178/eng/QAT1.7.L.4.14.0-00031.tar.gz
OpenSSL	openssl-3.0.0	https://github.com/openssl/openssl https://www.openssl.org/source/
Intel QAT Engine for OpenSSL	v0.6.7	https://github.com/intel/QAT_Engine
Intel ipsec-mb	1.00	https://github.com/intel/intel-ipsec-mb

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Intel® SGX DCAP Drivers	1.41	https://download.01.org/intel-sgx/sgx-dcap/1.10.3/linux/
Intel® SGX SDK	2.14.100.2	https://download.01.org/intel-sgx/sgx-dcap/1.10.3/linux/
Intel® KMRA AppHSM	1.2.1	https://hub.docker.com/r/intel/kmra
Intel® KMRA CTK	1.2.1	https://hub.docker.com/r/intel/kmra
Intel® KMRA PCCS	1.2.1	https://hub.docker.com/r/intel/kmra

7 Post Deployment Verification Guidelines

This section describes a set of processes that you can use to verify the components deployed by the scripts. The processes are not Configuration Profile-specific. They can be implemented for each of the Configuration Profiles described in the following appendixes:

- Appendix B, BMRA Basic Configuration Profile Setup
- Appendix C, BMRA Full Configuration Profile Setup
- Appendix D, BMRA On-Premises Edge Configuration Profile Setup
- Appendix E, BMRA Remote CO-Forwarding Configuration Profile Setup
- Appendix F, BMRA Regional Data Center Configuration Profile Setup

7.1 Check the Kubernetes Cluster

Perform the following steps:

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

```
# kubectl get nodes -o wide
NAME          STATUS   ROLES    AGE    VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION     CONTAINER-RUNTIME
controller1   Ready    master   4d1h   v1.19.8   10.250.192.155   <none>        Ubuntu 20.04.2
LTS   5.4.0-66-generic   docker://19.3.12
node1         Ready    <none>   4d1h   v1.19.8   10.250.190.112   <none>        Ubuntu 20.04.2
LTS   5.4.0-66-generic   docker://19.3.12
```

2. Check pod status of control nodes and worker nodes. All pods should be in Running or Completed status.

```
# kubectl get pods --all-namespaces
NAMESPACE                NAME                                                READY
STATUS      RESTARTS   AGE
cert-manager             cert-manager-56b686b465-g44nj                       1/1
Running     0          41h
cert-manager             cert-manager-cainjector-75c94654d-pvl7x             1/1
Running     14         41h
cert-manager             cert-manager-webhook-69bd5c9d75-tldnf               1/1
Running     8          41h
istio-operator           istio-operator-77bb9d9884-stf4m                     1/1
Running     0          40h
istio-system             istio-ingressgateway-574dff7b88-98zzp               1/1
Running     0          40h
istio-system             istiod-59db6b6d9-cdbmd                              1/1
Running     0          40h
kmra                     kmra-apphsm-6d7d945d54-nj9zc                        2/2
Running     0          40h
kmra                     kmra-ctk-55d989fc6f-ftc6n                           1/1
Running     0          40h
kmra                     kmra-pccs-b744d7d99-hwjmj                           2/2
Running     0          40h
kube-system              calico-kube-controllers-5b4d7b4594-rnn8h            1/1
Running     2          41h
kube-system              calico-node-2jbxq                                   1/1
Running     3          41h
```

```
kube-system              calico-node-tw8c5                                          1/1
Running     0          41h
kube-system              cmk-init-discover-ar09-03-cyp-pp7pb                        0/3
Completed   0          40h
kube-system              cmk-rfckb                                                  2/2   Running
0          40h
kube-system              cmk-webhook-6c9d5f8578-8cqb4                               1/1
Running     0          40h
kube-system              container-registry-7869d9c577-p9bs4                        2/2
Running     0          41h
kube-system              coredns-8474476ff8-4wm8f                                   1/1
Running     1          41h
kube-system              coredns-8474476ff8-rprlm                                   1/1
Running     1          41h
kube-system              dns-autoscaler-7df78bfcfb-8zxdz                            1/1
Running     1          41h
kube-system              intel-qat-plugin-intel-qat-plugin-htb48                    1/1
Running     0          40h
kube-system              intel-sgx-aesmd-9n68d                                      1/1
Running     0          40h
kube-system              intel-sgx-plugin-zqxp9-vp7bp                               1/1
Running     0          40h
kube-system              inteldeviceplugins-controller-manager-577b89579c-grdgx     2/2
Running     0          40h
kube-system              kube-apiserver-ar09-17-cyp                                 1/1
Running     0          41h
kube-system              kube-controller-manager-ar09-17-cyp                        1/1
Running     2          41h
kube-system              kube-multus-ds-amd64-5b5vs                                 1/1
Running     1          41h
kube-system              kube-multus-ds-amd64-chp98                                 1/1
Running     1          41h
kube-system              kube-proxy-6lsbf                                           1/1
Running     2          41h
kube-system              kube-proxy-cg62f                                           1/1
Running     2          41h
kube-system              kube-scheduler-ar09-17-cyp                                 1/1
Running     0          40h
kube-system              kubernetes-dashboard-785dcbb76d-fjbkg                      1/1
Running     1          41h
kube-system              kubernetes-metrics-scraper-5558854cb-flh56                 1/1
Running     1          41h
kube-system              nginx-proxy-ar09-03-cyp                                    1/1
Running     2          41h
kube-system              node-feature-discovery-controller-67b59d6cf4-7gnvv         1/1
Running     0          40h
kube-system              node-feature-discovery-worker-8d9mp                        1/1
Running     0          40h
monitoring               node-exporter-2stqw                                        2/2
Running     0          40h
monitoring               node-exporter-w4tfm                                        2/2
Running     0          40h
monitoring               prometheus-k8s-0                                           4/4
Running     0          40h
monitoring               prometheus-operator-bf54b8f56-fhj78                        2/2
Running     0          40h
monitoring               tas-telemetry-aware-scheduling-84ff454dfb-c86c7            1/1
Running     0          40h
monitoring               telegraf-tpj2z                                             2/2
Running     0          40h
sriov-network-operator   sriov-device-plugin-zs226                                  1/1
Running     0          40h
sriov-network-operator   sriov-network-config-daemon-cwbsm                          3/3
Running     0          40h
sriov-network-operator   sriov-network-operator-bb8ff65d9-2td74                     1/1
Running     0          40h
```

7.2 Check Intel Speed Select Technology – Base Frequency (Intel SST-BF) Configuration on 2nd Generation Intel Xeon Scalable Processor

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 2700/2800 minimum and 2700/2800 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, as shown below with mode `s`.

```
Name = 6252N
CPUs = 96
Base = 2300
     |------sysfs-------|
Core | base   max   min |
-----|------------------|
   0 | 2100  2100  2100 |
   1 | 2800  2800  2800 |
   2 | 2800  2800  2800 |
   3 | 2100  2100  2100 |
(...)
  94 | 2800  2800  2800 |
  95 | 2100  2100  2100 |
-----|------------------|
```

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

Note: The minimum OS distribution with Intel SST-BF support is Ubuntu 20.04 and CentOS 8.2.

7.3 Check Intel Speed Select Technology on 3rd Generation Intel Xeon Scalable Processor

The steps in this section describe how to verify Intel Speed Select Technology.

7.3.1 Check Intel Speed Select Technology - Base Frequency (Intel SST-BF) Configuration

To display Intel SST-BF properties, use the following command:

```
root@sdp1146:~# intel-speed-select base-freq info -l 0
Intel® Speed Select Technology
Executing on CPU model:106[0x6a]
 package-0
  die-0
    cpu-0
      speed-select-base-freq-properties
        high-priority-base-frequency(MHz):2400
        high-priority-cpu-mask:00000000,000030cc,cc0c0330
        high-priority-cpu-list:4,5,8,9,18,19,26,27,30,31,34,35,38,39,44,45
        low-priority-base-frequency(MHz):1800
        tjunction-temperature(C):105
        thermal-design-power(W):185
 package-1
  die-0
    cpu-48
      speed-select-base-freq-properties
        high-priority-base-frequency(MHz):2400
        high-priority-cpu-mask:000c0f3c,c3c00000,00000000
        high-priority-cpu-list:54,55,56,57,62,63,66,67,68,69,72,73,74,75,82,83
        low-priority-base-frequency(MHz):1800
        tjunction-temperature(C):105
        thermal-design-power(W):185
```

To help ensure that Intel SST-BF is enabled, check the CPU base frequency:

```
root@sdp1146:~# cat /sys/devices/system/cpu/cpu0/cpufreq/base_frequency
1800000
root@sdp1146:~# cat /sys/devices/system/cpu/cpu4/cpufreq/base_frequency
```

```
2400000
root@sdp1146:~# cat /sys/devices/system/cpu/cpu5/cpufreq/base_frequency
2400000
```

CPUs 4 and 5 are marked as high priority CPUs and have base frequency of 2.4 GHz.

CPU 0 is marked as low priority CPU and has base frequency of 1.8 GHz.

7.3.2 Check Intel Speed Select Technology – Core Power (Intel SST-CP)

Intel SST-CP enables a user to set up to four Classes of Service (CLOS) and assign CPUs to certain CLOSes.

To verify the correctness of CLOS 0, use the following command:

```
root@sdp1146:~# intel-speed-select core-power get-config -c 0
Intel® Speed Select Technology
Executing on CPU model:106[0x6a]
 package-0
  die-0
    cpu-0
      core-power
        clos:0
        epp:0
        clos-proportional-priority:0
        clos-min:2400 MHz
        clos-max:Max Turbo frequency
        clos-desired:0 MHz
 package-1
  die-0
    cpu-48
      core-power
        clos:0
        epp:0
        clos-proportional-priority:0
        clos-min:2400 MHz
        clos-max:Max Turbo frequency
        clos-desired:0 MHz
root@sdp1146:~/linux/tools/power/x86/intel-speed-select#
```

To verify the CLOS assignment for CPUs 0, 4, and 5, use the following command:

```
root@sdp1146:~# intel-speed-select -c 0,4-5 core-power get-assoc
Intel® Speed Select Technology
Executing on CPU model:106[0x6a]
 package-0
  die-0
    cpu-0
      get-assoc
        clos:3
 package-0
  die-0
    cpu-4
      get-assoc
        clos:0
 package-0
  die-0
    cpu-5
      get-assoc
        clos:0
root@sdp1146:~/linux/tools/power/x86/intel-speed-select#
```

To learn more about Intel SST-CP or Intel SST-CP Classes of Service, refer to: https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html#intel-r-speed-select-technology-core-power-intel-r-sst-cp

7.4 Check Intel Speed Select Technology – Performance Profile (Intel SST-PP) with Intel Speed Select Technology – Turbo Frequency (Intel SST-TF) on 3rd Generation Intel Xeon Scalable Processors

To verify the availability of Intel SST-PP and its features, use the following command.

```
root@sdp1146:~/linux/tools/power/x86/intel-speed-select#
[root@as09-35-ac ~]# intel-speed-select –info
Intel(R) Speed Select Technology
```

```
Executing on CPU model:143[0x8f]
Platform: API version : 1
Platform: Driver version : 1
Platform: mbox supported : 1
Platform: mmio supported : 1
Intel(R) SST-PP (feature perf-profile) is supported
TDP level change control is unlocked, max level: 3
Intel(R) SST-TF (feature turbo-freq) is supported
Intel(R) SST-BF (feature base-freq) is supported
Intel(R) SST-CP (feature core-power) is supported
```

To verify that Intel SST-PP is unlocked in the BIOS, use the following command:

```
[root@as09-35-ac ~]# intel-speed-select perf-profile get-lock-status
Intel(R) Speed Select Technology
Executing on CPU model:143[0x8f]
Caching topology information
 package-0
  die-0
    cpu-0
      get-lock-status:unlocked
```

To confirm that the statuses of Intel SST-BF, Intel SST-CP, and Intel SST-TF are `enabled` or `disabled`, which must match the value of SST-PP in `host_vars`, and to check the properties of `perf-level`, use the following commands:

```
[root@as09-35-ac ~]# intel-speed-select perf-profile info -l 0 2>&1 | grep 'speed-select'
        speed-select-turbo-freq:enabled
        speed-select-base-freq:enabled
        speed-select-core-power:enabled
        speed-select-base-freq-properties
        speed-select-turbo-freq-properties
          speed-select-turbo-freq-clip-frequencies

[root@as09-35-ac ~]# intel-speed-select perf-profile get-config-levels
Intel(R) Speed Select Technology
Executing on CPU model:143[0x8f]
 package-0
  die-0
    cpu-0
      get-config-levels:3
```

Note: `config-levels` must be `get-config-levels:3` (Intel SST-BF, Intel SST-CP, and Intel SST-TF). `get-config-levels: 0` means misconfiguration of setup in software or BIOS.

Set turbo status `on` and `off` to verify busy workload frequency ranges dynamically when Intel SST-PP is configured. For example, if CPUs 0,40,1,41,2,42,3,43,5,45,8,48,9,49,10,50,11,51,13,53 are defined in `host_vars` for Intel SST-PP to get 100 MHz boost, use following commands for verification.

```
[root@as09-35-ac ~]# echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
[root@as09-35-ac ~]# turbostat -c 0,40,1,41,2,42,3,43,5,45,8,48,9,49,10,50,11,51,13,53 --show Package,Core,CPU,Bzy_MHz -i
1

Core    CPU     Bzy_MHz
-       -       798
0       0       800
0       40      803
1       1       800
1       41      799
2       2       800
2       42      803
3       3       800
3       43      803
5       5       800
5       45      801
8       8       800
8       48      803
9       9       800
9       49      802
10      10      800
```

```
10          50          798
11          11          800
11          51          802
13          13          800
13          53          800
```

```
[root@as09-35-ac ~]# echo 0 > /sys/devices/system/cpu/intel_pstate/no_turbo
[root@as09-35-ac ~]# turbostat -c 0,40,1,41,2,42,3,43,5,45,8,48,9,49,10,50,11,51,13,53 --show Package,Core,CPU,Bzy_MHz -i
1

Core        CPU     Bzy_MHz
-           -       2888
0           0       2896
0           40      2885
1           1       2884
1           41      2901
2           2       2899
2           42      2888
3           3       2900
3           43      2895
5           5       2889
5           45      2900
8           8       2901
8           48      2898
9           9       2857
9           49      2900
10          10      2900
10          50      2838
11          11      2900
11          51      2900
13          13      2900
13          53      2902
14          14      2901
```

Note that improved performance in a frequency range can be observed dynamically as it jumps from approximately 800 to approximately 2900 in CPUs when Intel SST-PP is configured with `turbo-freq`.[14]

7.5 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 runtime, without resetting or rebooting the server.

7.5.1 Check DDP Profiles in Intel® Ethernet 700 Series Network Adapters

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

```
~/ddp-tool# ./ddptool -a
Intel(R) Dynamic Device Personalization Tool
DDPTool version 2020.17.22.7
Copyright (C) 2019 - 2021 Intel Corporation.

NIC  DevId D:B:S.F      DevName         TrackId  Version      Name
==== ===== ============ =============== ======== ============ ==============================
001) 158B  0000:18:00.0 ens785f0        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
002) 158B  0000:18:00.1 ens785f1        80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
003) 154C  0000:18:02.0 ens785f0v0      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
004) 154C  0000:18:02.1 ens785f0v1      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
005) 154C  0000:18:02.2 ens785f0v2      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
006) 154C  0000:18:02.3 ens785f0v3      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
007) 154C  0000:18:02.4 ens785f0v4      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
```

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

```
008) 154C  0000:18:02.5 ens785f0v5      80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
009) 154C  0000:18:0A.0 N/A             80000008 1.0.3.0      GTPv1-C/U IPv4/IPv6 payload
010) 154C  0000:18:0A.1 N/A             -        -
011) 154C  0000:18:0A.2 N/A             -        -
012) 154C  0000:18:0A.3 N/A             -        -
013) 1592  0000:AF:00.0 ens801f0        -        -            -
014) 1592  0000:AF:00.1 ens801f1        -        -            -
```

Download `ddptool` at: https://github.com/intel/ddp-tool/tree/1.0.9.0.

7.5.2 Check DDP Profiles in Intel® Ethernet 800 Series Network Adapters

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

```
~/ddp-tool# ./ddptool -a
Intel(R) Dynamic Device Personalization Tool
DDPTool version 2020.17.22.7
Copyright (C) 2019 - 2021 Intel Corporation.

NIC  DevId D:B:S.F      DevName         TrackId  Version      Name
==== ===== ============ =============== ======== ============ ==============================
001) 1592  0000:B1:00.0 ens801f0        C0000002 1.3.30.0     ICE COMMS Package
002) 1592  0000:B1:00.1 ens801f1        C0000002 1.3.30.0     ICE COMMS Package
003) 1889  0000:B1:01.0 ens801f0v0      C0000002 1.3.30.0     ICE COMMS Package
004) 1889  0000:B1:01.1 ens801f0v1      C0000002 1.3.30.0     ICE COMMS Package
005) 1889  0000:B1:01.2 ens801f0v2      C0000002 1.3.30.0     ICE COMMS Package
006) 1889  0000:B1:01.3 ens801f0v3      C0000002 1.3.30.0     ICE COMMS Package
007) 1889  0000:B1:01.4 ens801f0v4      C0000002 1.3.30.0     ICE COMMS Package
008) 1889  0000:B1:01.5 ens801f0v5      C0000002 1.3.30.0     ICE COMMS Package
009) 1889  0000:B1:11.0 N/A             C0000002 1.3.30.0     ICE COMMS Package
010) 1889  0000:B1:11.1 N/A             -        -
011) 1889  0000:B1:11.2 N/A             -        -
012) 1889  0000:B1:11.3 N/A             -        -
```

Download `ddptool` at: https://github.com/intel/ddp-tool/tree/1.0.9.0.

7.5.3 Check SR-IOV Resources

After everything is installed and set up correctly, you see that the device plugin is able to discover VFs with 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/intel_sriov_dpdk_700_series": "2",
  "intel.com/intel_sriov_dpdk_800_series": "0",
  "memory": "7880620Ki",
  "pods": "100"
}
```

BMRA does not create SR-IOV resources with DDP by default. These must be created by the user, as described in the Dynamic Device Personalization section.

7.6 Check Node Feature Discovery

Node Feature Discovery (NFD) is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform that can, in turn, be used to facilitate intelligent scheduling of a workload. 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 publishes these as node labels using the Kubernetes API. NFD only handles non-allocatable features.

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

```
# kubectl get ds --all-namespaces | grep node-feature-discovery
kube-system   node-feature-discovery-worker    1    1    1    1    1    <none>    3d2h
```

To check the labels created by NFD, use the following command:

```
# kubectl label node --list --all
Listing labels for Node./controller1:
 kubernetes.io/arch=amd64
 kubernetes.io/hostname=controller1
 kubernetes.io/os=linux
 node-role.kubernetes.io/master=
 beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
Listing labels for Node./node1:
 beta.kubernetes.io/arch=amd64
 feature.node.kubernetes.io/cpu-pstate.turbo=true
 feature.node.kubernetes.io/cpu-cpuid.VMX=true
 feature.node.kubernetes.io/kernel-version.minor=4
 feature.node.kubernetes.io/cpu-rdt.RDTMBM=true
 feature.node.kubernetes.io/cpu-cpuid.AESNI=true
 feature.node.kubernetes.io/cpu-cpuid.AVX512F=true
 feature.node.kubernetes.io/pci-0b40_8086.present=true
 feature.node.kubernetes.io/memory-numa=true
 feature.node.kubernetes.io/cpu-cpuid.MPX=true
 kubernetes.io/os=linux
 feature.node.kubernetes.io/network-sriov.configured=true
 feature.node.kubernetes.io/cpu-power.sst_bf.enabled=true
 feature.node.kubernetes.io/system-os_release.VERSION_ID.minor=04
 feature.node.kubernetes.io/cpu-cpuid.ADX=true
 feature.node.kubernetes.io/cpu-cpuid.AVX512VL=true
 feature.node.kubernetes.io/cpu-rdt.RDTMON=true
 feature.node.kubernetes.io/system-os_release.VERSION_ID.major=20
 feature.node.kubernetes.io/cpu-cpuid.AVX512VNNI=true
 feature.node.kubernetes.io/system-os_release.VERSION_ID=20.04
 feature.node.kubernetes.io/kernel-version.revision=0
 feature.node.kubernetes.io/pci-0300_1a03.present=true
 kubernetes.io/arch=amd64
 feature.node.kubernetes.io/cpu-cpuid.AVX512CD=true
 feature.node.kubernetes.io/kernel-version.major=5
 kubernetes.io/hostname=node1
 feature.node.kubernetes.io/network-sriov.capable=true
 feature.node.kubernetes.io/cpu-cpuid.AVX512DQ=true
 feature.node.kubernetes.io/cpu-cpuid.IBPB=true
 feature.node.kubernetes.io/system-os_release.ID=ubuntu
 feature.node.kubernetes.io/cpu-cpuid.STIBP=true
 feature.node.kubernetes.io/cpu-rdt.RDTL3CA=true
 feature.node.kubernetes.io/kernel-config.NO_HZ_IDLE=true
 feature.node.kubernetes.io/cpu-rdt.RDTMBA=true
 feature.node.kubernetes.io/storage-nonrotationaldisk=true
 feature.node.kubernetes.io/iommu-enabled=true
 feature.node.kubernetes.io/cpu-cpuid.AVX2=true
 feature.node.kubernetes.io/cpu-cpuid.FMA3=true
 feature.node.kubernetes.io/cpu-cpuid.AVX=true
 feature.node.kubernetes.io/kernel-config.NO_HZ=true
 beta.kubernetes.io/os=linux
 feature.node.kubernetes.io/cpu-hardware_multithreading=true
 feature.node.kubernetes.io/cpu-rdt.RDTCMT=true
 feature.node.kubernetes.io/kernel-version.full=5.4.0-66-generic
 cmk.intel.com/cmk-node=true
 feature.node.kubernetes.io/cpu-cpuid.AVX512BW=true
```

The node labels can be used when provisioning a pod. In the following example, the pod is scheduled only if there is a node with CPU Manager for Kubernetes (CMK) available:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nfd-1
spec:
  nodeSelector:
    cmk.intel.com/cmk-node: "true"
  containers:
  - name: pod-nfd-1
    image: ubuntu:focal
    command:
```

```
    - "/bin/bash"
    - "-c"
    args:
    - "tail -f /dev/null
```

In the above node labels, this is true for `node1`, and the pod is scheduled there. If no node is available that matches the nodeSelectors, the pod remains in `pending` status.

7.7 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 treats 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 help ensure that a CPU (or set of CPUs) is isolated from the kernel scheduler. Then the latency-sensitive workload process threads 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 has 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 receives the requested resources provided they are available

CPU Manager for Kubernetes creates 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 a successful CPU Manager for Kubernetes deployment and CPU initialization. In the example setup, Intel HT Technology is enabled; therefore, both physical and associated logical processors are isolated. Using the default values in the `group_vars/all.yml` file, CPU Manager for Kubernetes allocates two cores to the exclusive pool, and two cores to the shared pool. The default values for isolcpus for the node are “4-11”, which only partially isolates the physical cores as the system is configured with Intel HT Technology. Inspecting the logs for CPU Manager for Kubernetes shows output similar to the following.

```
root@av09-07-wp:~# kubectl get pods -A
NAMESPACE     NAME                                      READY   STATUS             RESTARTS   AGE
kube-system   cmk-init-discover-av09-06-wp-pl5tv        0/3     Completed          0          26m
kube-system   cmk-rnq9q                                 2/2     Running            0          25m
kube-system   cmk-webhook-6c9d5f8578-jkfc9              1/1     Running            0          25m
```

```
root@av09-07-wp:~# kubectl get cm
NAME                    DATA   AGE
cmk-config-av09-06-wp   1      30m

root@av09-07-wp:~# kubectl describe cm cmk-config-av09-06-wp
Name:         cmk-config-av09-06-wp
Namespace:    default
Labels:       <none>
Annotations:  Owner:

Data
====
config:
----
exclusive:
  0:
    0,44: []
    1,45: []
```

```
 1: {}
infra:
  0:

4,48,5,49,6,50,7,51,8,52,9,53,10,54,11,55,12,56,13,57,14,58,15,59,16,60,17,61,18,62,19,63,20,64
,21,65:
    - '40218'
  1:
    ?
22,66,23,67,24,68,25,69,26,70,27,71,28,72,29,73,30,74,31,75,32,76,33,77,34,78,35,79,36,80,37,81
,38,82,39,83,40,84,41,85,42,86,43,87
    : - '40167'
shared:
  0:
    2,46,3,47: []
  1: {}

Events:  <none>
```

CPU Manager for Kubernetes prioritizes using high priority Intel SST-BF cores for the exclusive core list when possible. If there are fully isolated physical cores (through isolcpus), these are also prioritized for the exclusive pool. In the above output, there are no physical cores that are fully isolated, in which case CPU Manager for Kubernetes only looks at high priority Intel SST-BF cores.

On successful run, the allocatable resource list for the node should be updated with resources 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 node1 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "2",
  "cpu": "93",
  "ephemeral-storage": "452220352993",
  "hugepages-1Gi": "4Gi",
  "intel.com/intel_sriov_dpdk_700_series": "2",
  "intel.com/intel_sriov_dpdk_800_series": "0",
  "intel.com/intel_sriov_netdevice": "4",
  "memory": "191733164Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
```

CPU Manager for Kubernetes helps ensure 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.

An example pod requesting one exclusive core can be seen below:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-cmk-1
  annotations:
    cmk.intel.com/mutate: "true"
  namespace: kube-system
spec:
  serviceAccountName: cmk
  containers:
  - name: pod-cmk-1
    image: ubuntu:focal
    command:
    - "/bin/bash"
    - "-c"
    args:
    - "tail -f /dev/null"
    resources:
      requests:
        cmk.intel.com/exclusive-cores: '1'
      limits:
        cmk.intel.com/exclusive-cores: '1'
    volumeMounts:
    - mountPath: /opt/bin
      name: cmk-install-dir
```

```
  volumes:
    - hostPath:
        path: /opt/bin
      name: cmk-install-dir
```

After creating the pod, the core allocation from CPU Manager for Kubernetes can be seen in the pod:

```
# kubectl exec pod-cmk-1 -n kube-system -- /opt/bin/cmk isolate --pool=exclusive env | grep CMK
CMK_CPUS_ASSIGNED_MASK=2000000000002
CMK_CPUS_ASSIGNED=1,49
CMK_PROC_FS=/host/proc
CMK_NUM_CORES=1
CMK_CPUS_INFRA=0,48,3,51,4,52,5,53,6,54,7,55,8,56,11,59,12,60,13,61,14,62,15,63,16,64,17,65,18,
66,19,67,20,68,21,69,22,70,23,71,24,72,25,73,26,74,27,75,28,76,29,77,30,78,31,79,32,80,33,81,34
,82,35,83,36,84,37,85,38,86,39,87,40,88,41,89,42,90,43,91,44,92,45,93,46,94,47,95
```

By default, CPU Manager for Kubernetes sets the affinity of the application. Some applications, e.g., DPDK-based ones, usually take the core allocation as an input parameter. These can be run by adding the `--no-affinity` option to CPU Manager for Kubernetes, and then reading the list of `CMK_CPUS_ASSIGNED` and using these for core pinning.

For more details on usage, see: https://github.com/intel/CPU-Manager-for-Kubernetes

7.8 Check Topology Manager

An increasing number of systems use 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 help extract the optimal performance[15], required optimizations related to CPU isolation, memory, and device locality must be made. It is enabled by default starting with Kubernetes 1.19.8. Topology Manager is a beta feature and is enabled by default.

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 stores this and admits 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 rejects this pod from the node. This results in a pod in a Terminated state with a pod admission failure.
  After 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 stores 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 rejects the pod from the node. This results in a pod in a Terminated state with a pod admission failure.
  After 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
```

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

```
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.
```

7.8.1 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:

```
...
# Enable Kubernetes built-in Topology Manager
topology_manager_enabled: true
# There are four supported policies: none, best-effort, restricted, single-numa-node.
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
```

7.8.2 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. Log in to the worker node via SSH, for example:

```
# ssh node1
```

2. Edit the kubelet configuration in the `/etc/kubernetes/kubelet-config.yaml` file:

```
(...)
topologyManagerPolicy: single-numa-node
(...)
```

3. Restart the Kubelet service:

```
# systemctl restart kubelet
```

7.9 Check Intel Device Plugins for Kubernetes

Like other vendors, Intel provides many hardware devices that help deliver efficient acceleration of graphics, computation, data processing, more security, and compression. Those devices optimize hardware for specific tasks, which saves CPU cycles for other workloads and typically results in performance gains.[16] The Kubernetes device plugin framework provides a vendor-independent solution for hardware devices. Intel has developed a set of device plugins that complies with the Kubernetes device plugin framework and allows users to request and consume hardware devices across Kubernetes clusters such as Intel® QuickAssist Technology, GPUs, and FPGAs. The detailed documentation and code are available at:

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

7.9.1 Check SR-IOV Network Device Plugin

The SR-IOV network device plugin discovers, filters, and exposes VFs on nodes in a cluster. The plugin creates consumable resources based on custom filters, which provides a flexible way of grouping VFs according to a set of selectors.

Using the default configuration, two resources are created. These resources both include VFs from common Intel network adapters but differentiate based on the driver used. Following a successful deployment, the resources are visible as shown below:

```
# kubectl get node node1 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "2",
  "cpu": "93",
  "ephemeral-storage": "452220352993",
  "hugepages-1Gi": "4Gi",
  "intel.com/intel_sriov_dpdk_700_series": "2",
  "intel.com/intel_sriov_dpdk_800_series": "0",
```

[16] Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products. See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

```
  "intel.com/intel_sriov_netdevice": "4",
  "memory": "191733164Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

In the above, there are two SR-IOV network device plugin resources: "`intel.com/intel_sriov_dpdk_710_series`" and "`intel.com/intel_sriov_netdevice`". The "netdevice" VFs are bound to a kernel driver and can be configured using the SR-IOV CNI Plugin. The "dpdk" VFs are bound to a DPDK driver, which allows an application to operate in userspace, bypassing the kernel network stack for ultra-high performance.[17]

To check allocation of a DPDK resource, create the following pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-sriov-dpdk-1
spec:
  containers:
  - name: pod-sriov-dpdk-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        intel.com/intel_sriov_dpdk_700_series: '1'
      limits:
        intel.com/intel_sriov_dpdk_700_series: '1'
```

After the pod is running, check that the resource is listed in the environment of the pod:

```
# kubectl exec pod-sriov-dpdk-1 -- env | grep PCIDEVICE
PCIDEVICE_INTEL_COM_INTEL_SRIOV_DPDK_700_SERIES=0000:86:02.1
```

The same approach can be used to check the "netdevice" resources, but if the default configuration has been used, the SR-IOV CNI and example network attachment definition were installed and can be used as well.

```
# kubectl get net-attach-def
NAME               AGE
sriov-net          3d23h
```

```
# kubectl describe net-attach-def sriov-net | grep Annotations
Annotations:  k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_netdevice
```

Using this information, create a pod that assigns a `netdevice` VF from SR-IOV network device plugin and creates an interface in the pod using the SR-IOV CNI plugin:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-sriov-netdevice-1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net
spec:
  containers:
  - name: pod-sriov-netdevice-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        intel.com/intel_sriov_netdevice: '1'
      limits:
        intel.com/intel_sriov_netdevice: '1'
```

Start by checking that the VF has been added to the pod, and then check that the interface has been created through SR-IOV CNI:

```
# kubectl exec pod-sriov-netdevice-1 -- env | grep PCIDEVICE
PCIDEVICE_INTEL_COM_INTEL_SRIOV_NETDEVICE=0000:86:0a.1
```

```
# kubectl exec pod-sriov-netdevice-1 -- ip a
```

[17] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if69: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether c6:0d:c3:a5:57:15 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.244.1.26/24 brd 10.244.1.255 scope global eth0
       valid_lft forever preferred_lft forever
16: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 7e:ad:44:41:91:e5 brd ff:ff:ff:ff:ff:ff
    inet 10.56.217.172/24 brd 10.56.217.255 scope global net1
       valid_lft forever preferred_lft forever
```

7.9.2 Check QAT Device Plugin

The Intel® QuickAssist Technology (Intel® QAT) device plugin discovers and exposes QAT device VFs as consumable resources in Kubernetes[18]. It works like the SR-IOV network device plugin but provides access to accelerated cryptographic and compression features.

If enabled and supported, QAT resources show up as a node resource:

```
# kubectl get node node1 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "2",
  "cpu": "93",
  "ephemeral-storage": "452220352993",
  "hugepages-1Gi": "4Gi",
  "intel.com/intel_sriov_dpdk_700_series": "2",
  "intel.com/intel_sriov_dpdk_800_series": "0",
  "intel.com/intel_sriov_netdevice": "4",
  "memory": "191733164Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

Now create a pod that requests a VF from the above resource:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-qat-1
spec:
  containers:
  - name: pod-qat-dpdk-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        qat.intel.com/generic: '1'
      limits:
        qat.intel.com/generic: '1'
```

After the pod is running, verify that the VF was correctly assigned to the pod:

```
kubectl exec pod-qat-1 -- env | grep QAT
QAT0=0000:3e:02.1
```

To further test the VFs, an application that supports offloading and acceleration is required, which can be done through DPDK. More information and examples can be found here: https://github.com/intel/intel-device-plugins-for-kubernetes/tree/master/demo

7.9.3 Check SGX Device Plugin

The Intel® SGX device plugin discovers and exposes SGX device nodes to kubelet as consumable resources in Kubernetes[19].

If enabled and supported, SGX resources show up as a node resource:

```
# kubectl get node node1 -o json | jq '.status.allocatable'
{
  "cmk.intel.com/exclusive-cores": "2",
  "cpu": "77",
```

[18] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

[19] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

```
  "ephemeral-storage": "282687233580",
  "hugepages-1Gi": "4Gi",
  "memory": "125651052Ki",
  "pods": "110",
  "sgx.intel.com/enclave": "20",
  "sgx.intel.com/epc": "1054863360",
  "sgx.intel.com/provision": "20"
}
```

7.10 Check Networking Features (After Installation)

This section describes how to verify certain CNI plugins.

7.10.1 Check Multus CNI Plugin

To verify that Multus CNI Plugin is running, an additional network can be created using the basic CNI plugins installed as part of the playbooks. For this example, the "macvlan" CNI is used. Start by creating a NetworkAttachmentDefinition (net-attach-def) using the provided template and update the `{{ interface }}` value to match an interface name on the worker nodes of the system, for example, "ens786f1".[20]

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-multus-1
spec:
  config: '{
            "cniVersion": "0.3.0",
            "type": "macvlan",
            "master": "{{ interface }}",
            "mode": "bridge",
            "ipam": {
                "type": "host-local",
                "ranges": [
                    [ {
                         "subnet": "10.10.0.0/16",
                         "rangeStart": "10.10.1.20",
                         "rangeEnd": "10.10.3.50",
                         "gateway": "10.10.0.254"
                    } ]
                ]
            }
        }'
```

After applying the configuration, verify that it has been created and is available in the cluster:

```
# kubectl get net-attach-def
NAME               AGE
macvlan-multus-1   4d1h
```

Following this, create a pod that requests an interface from the newly created `net-attach-def`:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-macvlan-1
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-multus-1
spec:
  containers:
  - name: pod-macvlan-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
```

After the pod is running, verify that the additional interface is available in the pod:

```
# kubectl exec pod-macvlan-1 -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

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

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if71: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether 7e:33:5e:5b:1b:4f brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.244.1.28/24 brd 10.244.1.255 scope global eth0
       valid_lft forever preferred_lft forever
4: net1@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 8e:c0:49:08:8a:ab brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.10.1.21/16 brd 10.10.255.255 scope global net1
       valid_lft forever preferred_lft forever
```

7.10.2 Check 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 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       4d3h
```

An example using the SR-IOV CNI Plugin can be found in Section 7.8.1. To test it again, create a pod as shown below:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-sriov-netdevice-1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net
spec:
  containers:
  - name: pod-sriov-netdevice-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        intel.com/intel_sriov_netdevice: '1'
      limits:
        intel.com/intel_sriov_netdevice: '1'
```

After the pod is running, verify that the additional network interface has been added to the pod:

```
# kubectl exec pod-sriov-netdevice-1 -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if72: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether 32:b8:e3:04:c8:93 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.244.1.29/24 brd 10.244.1.255 scope global eth0
       valid_lft forever preferred_lft forever
14: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether de:05:9d:bc:62:f3 brd ff:ff:ff:ff:ff:ff
    inet 10.56.217.173/24 brd 10.56.217.255 scope global net1
       valid_lft forever preferred_lft forever
```

7.10.3 Check Userspace CNI Plugin

The Userspace CNI is a Container Network Interface (CNI) plugin designed to implement userspace 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.[21]

By default, OVS is installed as the vSwitch, alongside a `net-attach-def` to expose vhostuser resources through Userspace CNI in Kubernetes.

Verify the resource is available using the following command:

[21] Refer to https://software.intel.com/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products. See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

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

With the `userspace-ovs` resource available, create a pod requesting an interface:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-userspace-1
  annotations:
    k8s.v1.cni.cncf.io/networks: userspace-ovs
spec:
  containers:
  - name: pod-userspace-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    volumeMounts:
    - mountPath: /vhu/
      name: socket
  volumes:
  - name: socket
    hostPath:
      path: /var/lib/cni/vhostuser/
```

After the pod is running, verify that the vhostuser socket has been added to the container:

```
# kubectl exec pod-userspace-1 -- ls /vhu/
e4c7e6fb63ec737f7aee3f451e79f7fba2cac6d212b88fb0725da1b9afed1cfb
```

Before checking OVS, check the node that the pod is deployed on:

```
# kubectl describe pod pod-userspace-1 | grep Node:
Node:         node1/<node IP>
```

Connect to the node using SSH, and check that the vhostuser socket and interface has been added to OVS:

```
# ovs-vsctl show
fbb7d4e6-6f93-4bcd-a254-ff10f1bd5fd7
    Bridge br0
        datapath_type: netdev
        Port br0
            Interface br0
                type: internal
        Port e4c7e6fb63ec-net1
            Interface e4c7e6fb63ec-net1
                type: dpdkvhostuser
    ovs_version: "2.13.0"
```

At this point, the vhostuser socket is ready to use in the pod. The steps for using VPP as the vSwitch are similar, but instead of the userspace CNI resource name `userspace-ovs`, it is `userspace-vpp`.

More details and examples can be found here: https://github.com/intel/userspace-cni-network-plugin

7.10.4 Check 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
```

Assuming the cluster was created using the provided configuration default, there should be SR-IOV network resources on the node as shown below:

```
# kubectl get node node1 -o json | jq '.status.allocatable'
  "cmk.intel.com/exclusive-cores": "2",
  "cpu": "93",
  "ephemeral-storage": "452220352993",
  "hugepages-1Gi": "4Gi",
```

```
  "intel.com/intel_sriov_dpdk_700_series": "2",
  "intel.com/intel_sriov_dpdk_800_series": "0",
  "intel.com/intel_sriov_netdevice": "4",
  "memory": "191733164Ki",
  "pods": "110",
  "qat.intel.com/generic": "32"
}
```

If there are `netdevice` VFs configured, create a simple SR-IOV CNI resource as shown below:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-bond-net
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_netdevice
spec:
  config: '{
  "type": "sriov",
  "name": "sriov-network",
  "spoofchk":"off"
}'
```

Now create a Bond CNI resource:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net
spec:
  config: '{
  "type": "bond",
  "cniVersion": "0.3.1",
  "name": "bond-net",
  "ifname": "bond0",
  "mode": "active-backup",
  "failOverMac": 1,
  "linksInContainer": true,
  "miimon": "100",
  "links": [
     {"name": "net1"},
     {"name": "net2"}
  ],
  "ipam": {
    "type": "host-local",
    "subnet": "10.56.217.0/24",
    "routes": [{
      "dst": "0.0.0.0/0"
    }],
    "gateway": "10.56.217.1"
  }
}'
```

Verify that both `bond-net` and `sriov-bond-net` were created:

```
# kubectl get net-attach-def
NAME             AGE
bond-net         15s
sriov-bond-net   36s
```

Now create a pod requesting two VFs from `sriov-bond-net`, which is used to create a bonded interface from `bond-net` through Bond CNI:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-bond-cni-1
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
{"name": "sriov-bond-net",
"interface": "net1"
},
{"name": "sriov-bond-net",
"interface": "net2"
```

```
},
{"name": "bond-net",
"interface": "bond0"
}]'
spec:
  containers:
  - name: pod-bond-cni-1
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        intel.com/intel_sriov_netdevice: '2'
      limits:
        intel.com/intel_sriov_netdevice: '2'
```

After the pod is running, verify that the two interfaces and the bonded interface were added using the command `ip a` inside the container. The two VFs are shown as `net1` and `net2`, and the bonded interface configured with an IP address is shown as `bond0`.

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

7.11 Check Grafana Telemetry Visualization

BMRA deploys Grafana for telemetry visualization. It is available on every cluster node on port 30000. Due to security reasons, this port is not exposed outside the cluster by default. Default credentials are admin/admin and you should change the default password after first login.

The Grafana TLS certificate is signed by the cluster CA and it is available in `/etc/kubernetes/ssl/ca.crt`

Visit Grafana at `https://<node-ip>:30000/`

BMRA comes with a set of dashboards from the kube-prometheus project (https://github.com/prometheus-operator/kube-prometheus). Dashboards are available in the Dashboards -> Manage menu as shown in Figure 12.

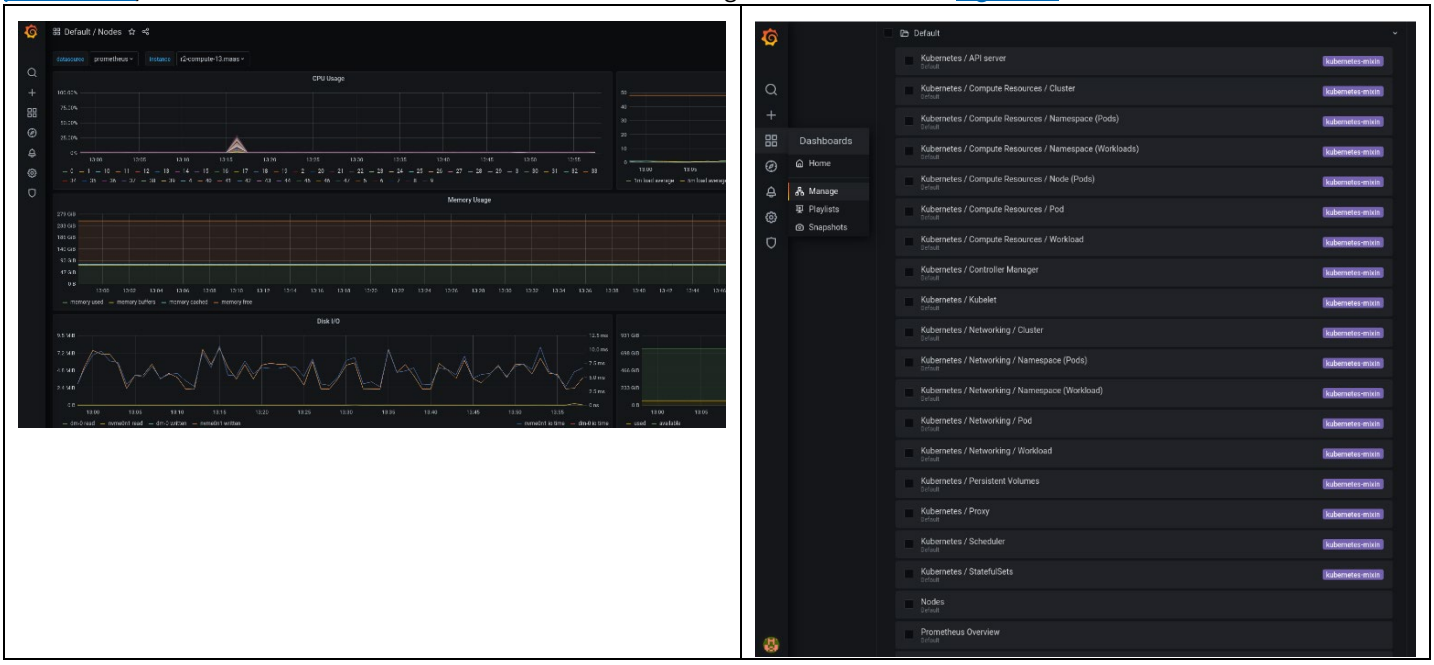

Figure 12. Grafana Dashboard Example

7.12 Check Telemetry Aware Scheduler

BMRA can deploy TAS Health Metric Demo Policy (https://github.com/intel/platform-aware-scheduling/blob/master/telemetry-aware-scheduling/docs/health-metric-example.md) when `tas_enable_demo_policy: true`, as shown below:

```
# Intel Telemetry Aware Scheduling
tas_enabled: true
tas_namespace: monitoring
```

```
# create and enable TAS demonstration policy: [true, false]
tas_enable_demo_policy: true
```

The Health Metric Demo Policy requires a Prometheus metric file to exist on the node and read by Prometheus. For security reasons, BMRA does not deploy it in the `/tmp` directory, where every user has access. Instead, it is deployed in the `/opt/intel/tas-demo-policy/` directory with root-only access.

To verify that the policy has been deployed, use the command:

```
kubectl get taspolicies -n monitoring
```

Details of this policy, including the rules and associated metrics, can be described with following command:

```
kubectl describe taspolicies demo-policy -n monitoring
```

To verify that the proper files exist on the worker node, use the following command:

```
# cat /opt/intel/tas-demo-policy/test.prom
node_health_metric 0
```

The node health metric value indicates the following:

- When `node_health_metric = 0`, it allows scheduling of pods on this node.
- When `node_health_metric = 2`, then the descheduler deschedules pods from this node.

7.12.1 Check Dontschedule Policy

To change the value of the `node_health_metric` to `dontschedule`, use the following command:

```
echo 'node_health_metric 2' | ssh <user@worker> -T "cat /opt/intel/tas-demo-policy/test.prom"
```

Then, to deploy a pod susceptible to the `dontschedule` policy, run the following command on the first controller node:

```
kubectl apply -f /usr/src/telemetry-aware-scheduling/deploy/health-metric-demo/demo-pod.yaml
```

The pod should not be deployed on node with `node_health_metric` == 2.

You can verify that TAS has been called to schedule the pod by looking at the logs:

```
kubectl logs pod/tas-telemetry-aware-scheduling-xxxx-yyyy -c tas-controller -n monitoring
```

Example output:

```
2019/08/19 15:30:59 NODE_B health_metric = 2
2019/08/19 15:30:59 NODE_A health_metric = 0
2019/08/19 15:30:59 NODE_A violating : health_metric Equals 2
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
```

7.12.2 Check Deschedule Policy

To see the impact of the descheduling policy, use a component called `descheduler`. For more details, visit https://github.com/intel/platform-aware-scheduling/blob/master/telemetry-aware-scheduling/docs/health-metric-example.md#seeing-the-impact

Set the `node_health_metric` to `deschedule` as follows:

```
echo 'node_health_metric 2' | ssh <user@worker> -T "cat /opt/intel/tas-demo-policy/test.prom"
```

Then run the descheduler with following command:

```
/usr/src/sigs.k8s.io/descheduler/_output/bin/descheduler --policy-config-file
/usr/src/telemetry-aware-scheduling/deploy/health-metric-demo/descheduler-policy.yaml --
kubeconfig /etc/kubernetes/admin.conf
```

The pod should be rescheduled onto a healthier node based on its TAS policy. If no other suitable node is available, the new pod fails to schedule.

7.13 Check Key Management Infrastructure with Intel SGX

To verify the Key Management infrastructure with SGX and use the private keys provisioned to Intel SGX enclaves, see Enabling Key Management NGINX Applications for step-by-step instructions to set up and run the NGINX workload.

7.14 Check Intel® Server GPU Device and Driver

BMRA deploys the intel-gpu/kernel project from GitHub for the latest Intel® Server GPU kernel driver for media processing. To verify that the devices and drivers are present in the system with the correct configuration, perform the following actions.

After installation, confirm the i915 driver presence and that the ASPEED device is ignored. The kernel option `915.force_probe=*` helps ensure the i915 driver probes for the SG1 device and `i915.enable_guc=2` helps ensure the device is enabled for SR-IOV.

```
# lsmod | grep i915
```

```
i915_spi               24576  0
mtd                    77824  17 i915_spi
i915                 2609152  0
video                  53248  1 i915
i2c_algo_bit           16384  1 i915
drm_kms_helper        217088  1 i915
drm                   610304  3 drm_kms_helper,i915

# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-5.4.48+ root=/dev/sda1 ro crashkernel=auto rhgb quiet
i915.force_probe=* modprobe.blacklist=ast,snd_hda_intel i915.enable_guc=2
```

Verify the Intel Server GPU devices (4907) are present and using i915 driver.

```
# lspci | grep -i VGA
02:00.0 VGA compatible controller: ASPEED Technology, Inc. ASPEED Graphics Family (rev 41)
1c:00.0 VGA compatible controller: Intel Corporation Device 4907 (rev 01)
21:00.0 VGA compatible controller: Intel Corporation Device 4907 (rev 01)
26:00.0 VGA compatible controller: Intel Corporation Device 4907 (rev 01)
2b:00.0 VGA compatible controller: Intel Corporation Device 4907 (rev 01)

# lspci -n -v -s 1c:00.0
1c:00.0 0300: 8086:4907 (rev 01) (prog-if 00 [VGA controller])
        Subsystem: 8086:35cf
        Flags: bus master, fast devsel, latency 0, IRQ 423, NUMA node 0
        Memory at a8000000 (64-bit, non-prefetchable) [size=16M]
        Memory at 387e00000000 (64-bit, prefetchable) [size=8G]
        Expansion ROM at <ignored> [disabled]
        Capabilities: [40] Vendor Specific Information: Len=0c <?>
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [ac] MSI: Enable+ Count=1/1 Maskable+ 64bit+
        Capabilities: [d0] Power Management version 3
        Capabilities: [100] Latency Tolerance Reporting
        Kernel driver in use: i915
        Kernel modules: i915
```

Confirm the Intel Server GPU kernel drivers are present on the system

```
# ls /dev/dri/ -l
total 0
crw-rw----. 1 root video 226,   0 Apr  8 10:15 card0
crw-rw----. 1 root video 226,   1 Apr  8 10:15 card1
crw-rw----. 1 root video 226,   2 Apr  8 10:15 card2
crw-rw----. 1 root video 226,   3 Apr  8 10:15 card3
crw-rw----. 1 root video 226, 128 Apr  8 10:15 renderD128
crw-rw----. 1 root video 226, 129 Apr  8 10:15 renderD129
crw-rw----. 1 root video 226, 130 Apr  8 10:15 renderD130
crw-rw----. 1 root video 226, 131 Apr  8 10:15 renderD131
```

You are now ready to deploy transcode workloads to utilize the hardware components. For more information, see the Open Visual Cloud GitHub site: https://github.com/OpenVisualCloud/CDN-Transcode-Sample.

7.15 Check Intel QAT Engine with OpenSSL

Check the version of Intel® QAT crypto engine present on the system.

```
# openssl engine -v qatengine
(qatengine) Reference implementation of QAT crypto engine(qat_sw) v0.6.7
     ENABLE_EXTERNAL_POLLING, POLL, ENABLE_HEURISTIC_POLLING,
     GET_NUM_REQUESTS_IN_FLIGHT, INIT_ENGINE
```

Run an OpenSSL speed test.

```
# openssl speed rsa2048
Doing 2048 bits private rsa's for 10s:
9943 2048 bits private RSA's in 9.99s
Doing 2048 bits public rsa's for 10s:
347087 2048 bits public RSA's in 10.00s
OpenSSL 1.1.1j  16 Feb 2021
built on: Thu Mar 25 15:44:30 2021 UTC
options:bn(64,64) rc4(16x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -ffile-
prefix-map=/build/openssl-8MEVD6/openssl-1.1.1j=. -flto=auto -ffat-lto-objects -fstack-
```

```
protector-strong -Wformat -Werror=format-security -DOPENSSL_TLS_SECURITY_LEVEL=2 -
DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -
DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -
DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -
DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.001005s 0.000029s    995.3  34708.7
```

Repeat the OpenSSL speed test with QAT engine for increased performance.

```
# openssl speed -engine qatengine -async_jobs 8 rsa2048
engine "qatengine" set.
Doing 2048 bits private rsa's for 10s:
45272 2048 bits private RSA's in 9.87s
Doing 2048 bits public rsa's for 10s:
756968 2048 bits public RSA's in 9.25s
OpenSSL 1.1.1j  16 Feb 2021
built on: Thu Mar 25 15:44:30 2021 UTC
options:bn(64,64) rc4(16x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -O2 -ffile-
prefix-map=/build/openssl-8MEVD6/openssl-1.1.1j=. -flto=auto -ffat-lto-objects -fstack-
protector-strong -Wformat -Werror=format-security -DOPENSSL_TLS_SECURITY_LEVEL=2 -
DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -
DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -
DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_ASM -DGHASH_ASM -
DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000218s 0.000012s   4586.8  81834.42632.7  48921.5
```

Note that the number of sign operations per second jumps from **995.3 to 4586.8**, which shows that approximately 19 to 20% performance gain is observed when Intel QAT Engine for OpenSSL is used.[22]

8 Conclusion – Automation Eases Reference Application Deployment

This document contains notes on installation, configuration, and use of networking and device plug-in 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 decrease 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 help 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 plug-ins in container-based NFV deployments.

[22] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Part 3:
Build Your Reference Architecture

Appendix A
BMRA Setup for All Configuration Profile Options

Appendix A BMRA Setup for All Configuration Profile Options

This appendix is relevant for generating BMRA Flavors based on their Configuration Profiles. It provides the prerequisites for a system setup and includes information that enables you to review BIOS prerequisites and software BOMs at a glance. The information is presented in multi-column tables to give an easy way to compare and assess the differences between the BMRA Flavors that are available.

After setting up the Kubernetes system, refer to the specific appendix from the following list to build the BMRA flavor:

Appendix B, BMRA Basic Configuration Profile Setup
Appendix C, BMRA Full Configuration Profile Setup
Appendix D, BMRA On-Premises Edge Configuration Profile Setup
Appendix E, BMRA Remote CO-Forwarding Configuration Profile Setup
Appendix F, BMRA Regional Data Center Configuration Profile Setup

Note: The taxonomy for the BMRA Configuration Profile settings is defined in Section 0.

A.1 Set Up an Ansible Host

BMRA Kubernetes clusters require an Ansible Host that stores information about all remote nodes managed. In general, any machine running a recent Linux distribution can be used as Ansible Host for any of the supported BMRA deployments (regardless of target OS on the control and worker nodes), as long as it meets the following basic requirements:

- Network connectivity to the control and worker nodes, including SSH
- Internet connection (using Proxy if necessary)
- Git utility installed
- Python 3 installed
- Ansible version 2.9.20 installed

Step-by-step instructions for building the Ansible Host are provided below for the same list of operating systems that are supported for the control and worker nodes (see Section 3.1.3):

A.1.1 CentOS Linux or RHEL Version 8 or Version 7 as Ansible Host

1. Install the Linux OS using any method supported by the vendor (CentOS Community or Red Hat, Inc., respectively). If using the iso image, choose the Minimal iso version, or select the "Minimal Install" (Basic functionality) option under Software Selection.
2. Make the proper configuration during installation for the following key elements: Network (Ethernet) port(s) IP Address; Host Name, Proxies (if necessary), and NTP (Network Time Protocol).
3. After the installation completes and the machine reboots, login as root and confirm that it has a valid IP address and can connect (ping) to the control and worker nodes.
4. Make sure the http and https proxies are set, if necessary, for internet access. The configuration can be completed with the *export* command or by including the following lines in the `/etc/environment` file:

   ```
   http_proxy=http://proxy.example.com:1080
   https_proxy=http://proxy.example.com:1080
   ```

   Then, load the proxies configuration in the current environment:

   ```
   # source /etc/environment
   ```

5. Install Git:

   ```
   # yum install -y git
   ```

6. Install Python 3:

   ```
   # yum -y install python3
   ```

7. Install Ansible:

   ```
   # pip3 install ansible==2.9.20
   ```

The Ansible Host box is now ready to deploy the Container BMRA. Follow the instructions in Section 3.3.

A.1.2 Ubuntu 20.04 LTS as Ansible Host

1. Install the OS using any method supported by the vendor (Canonical Ltd.). Either the Desktop or Server distribution can be used. Select the “Minimal installation” option under “Updates and Other software”.
2. Follow steps 2, 3, and 4 as described above for CentOS or RHEL.
3. Update the installation:

   ```
   # sudo apt update
   ```

4. Install SSH utilities:

```
# sudo apt install openssh-server
```

5. Install Git:
```
# sudo apt install -y git
```
6. Install Python 3-pip:
```
# sudo apt install -y python3-pip
```
7. Install Ansible:
```
# sudo pip3 install ansible==2.9.20
```

The Ansible Host box is now ready to deploy the Container BMRA. Follow the instructions in Section 3.3.

A.2 Set Up the Control and Worker Nodes - BIOS Prerequisites

This section is applicable for all **BMRA Configuration Profiles.**

Enter the UEFI or BIOS menu and update the configuration as shown in Table 23 and Table 24.

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 23. BIOS Prerequisites for Control and Worker Nodes for Basic and Full Configuration Profiles

PROFILES	BASIC CONFIGURATION PROFILE	FULL CONFIGURATION PROFILE
Configuration		
BIOS Profile	Energy Balance	Max Performance
Grub Command Line (values are set by Ansible)		
Isolcpus	Optional	Yes
Hugepages	Optional	Yes
P-state=disable	Optional	Yes, No-SST-BF
Limit C-state	Optional	Yes

Table 24. BIOS Prerequisites for Control and Worker Nodes for On-Premises Edge, Remote Co-Forwarding, and Regional Data Center Configuration Profiles

PROFILES	ON-PREMISES EDGE CONFIGURATION PROFILE	REMOTE CO-FORWARDING CONFIGURATION PROFILE	REGIONAL DATA CENTER CONFIGURATION PROFILE
Configuration			
BIOS Profile	Max Performance	Deterministic	Max Performance
Grub Command Line (values are set by Ansible)			
Isolcpus	Yes	Yes	Optional
Hugepages	Yes	Yes	Optional
P-state=disable	No	Yes, No-SST-BF	Optional
Limit C-state	No	Yes	Optional

The BIOS profile referenced in these tables consists of a number of configurations in the power management, thermal management, and configuration for Intel® platform technologies such as Intel® Virtualization Technology, Intel® Hyper-Threading Technology, Intel SpeedStep® technology, and Intel® Turbo Boost Technology.

The table provides three different BIOS profiles.
1. Energy Balance
2. Max Performance
3. Deterministic

The configuration and values set per each BIOS profile are defined in Table 18 and Table 19.

Note: The above values are the recommended configuration options on the Intel® S2600WFQ and Intel® M50CYP server boards. Some server boards may not provide the same options that are documented in this table. Vendors typically provide options for max performance configuration with virtualization.

A.3 Configuration Dictionary - Group Variables

All of the variables are important but pay special attention to the variables in **bold** as they almost always need to be updated to match the target environment.

Table 25. Configuration Dictionary – Group Variables

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
Common Cluster Configuration				
Kubernetes		Boolean	true/false	Specifies whether to deploy Kubernetes
	kube_version	String	v1.21.1	Kubernetes version
	container_runtime	String	docker, crio, containerd	Container runtime to use as base engine for cluster deployment
	docker_version	String	19.03	Docker version
	containerd_version	String	1.4.6	Containerd version
	crio_version	String	1.21.3	CRI-O version
	update_all_packages	Boolean	false	Runs system-wide package update (apt dist-upgrade, yum update, ...). Tip: Can be set using host_vars for more granular control.
	http_proxy	**URL**	**http://proxy.example.com:1080**	**HTTP proxy address. Comment out if your cluster is not behind proxy.**
	https_proxy	**URL**	**http://proxy.example.com:1080**	**HTTPS proxy address. Comment out if your cluster is not behind proxy.**
	additional_no_proxy	**Comma-separated list of addresses**	**.example.com**	**Additional URLs that are not behind proxy, for example your corporate intra network DNS domain, e.g., ".intel.com". Note: Kubernetes nodes addresses, pod network, etc. are added to no_proxy automatically.**
	kube_network_plugin_multus	Boolean	True	Specifies whether to use the network plugin Multus
	multus_version	String	V3.7	Multus version
	kube_network_plugin	String	calico/flannel	Specifies networking CNI to use
	kube_pods_subnet	CIDR	10.244.0.0/16	Kubernetes pod subnet. Make sure that it matches your CNI plugin requirements (Calico by default) and doesn't overlap with your corporate LAN.
	kube_service_addresses	CIDR	10.233.0.0/18	Kubernetes service subnet. Make sure that it matches your CNI plugin requirements (Calico by default) and doesn't overlap with your corporate LAN.
	kube_proxy_mode	String	Iptables	Instructs kube_proxy how to set up NAT and load balancing functions
	kube_proxy_nodeport_addresses_cidr:	CIDR	127.0.0.0/8	Kubernetes service subnet
	cluster_name	DNS domain	cluster.local	Name of the cluster
	registry_local_address	String	"localhost:30500"	Container registry address IP and port
	psp_enabled	Boolean	true/false	Enable pod security policy admission controller and create minimal set of rules
	always_pull_enabled	Boolean	true/false	Set image pull policy to Always. Pulls images before starting containers. Valid credentials must be configured.
Node Feature Discovery				

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
nfd_enabled		Boolean	true/false	Specifies whether to deploy Node Feature Discovery
	nfd_version	String	0.9	NFD version
	nfd_build_image_locally	Boolean	false	Builds NFD image locally instead of using the one from public registry.
	nfd_namespace	String	kube-system	Kubernetes namespace used for NFD deployment
	nfd_sleep_interval	String	60s	Defines how often NFD queries node status and update node labels
Intel CPU Manager for Kubernetes[23]				
cmk_enabled		Boolean	true/false	Enables Intel CPU Manager for Kubernetes.
	cmk_namespace	String	kube-system	Kubernetes namespace used for CMK deployment
	cmk_use_all_hosts	Boolean	false	Enables all hosts
	cmk_hosts_list	**Comma-separated strings**	**node1,node2**	**Comma-separated list of K8s worker node names that the CMK runs on**
	cmk_shared_num_cores	**Integer**	**2**	**Number of CPU cores to be assigned to the "shared" pool on each of the nodes**
	cmk_exclusive_num_cores	**Integer**	**2**	**Number of CPU cores to be assigned to the "exclusive" pool on each of the nodes**
	cmk_shared_mode	String, options: packed, spread	packed	Shared pool allocation mode
	cmk_exclusive_mode	String, options: packed, spread	packed	Exclusive pool allocation mode
Native Built-in Kubernetes CPU Manager				
native_cpu_manager_enabled		Boolean	true/false	Enabling CMK and built-in CPU Manager is not recommended. Setting this option as "true" enables the "static" policy; otherwise the default "none" policy is used.
	native_cpu_manager_system_reserved_cpus	Kubernetes millicores	2000m	Number of CPU cores to be reserved for housekeeping (2000m = 2000 millicores = 2 cores)
	native_cpu_manager_kube_reserved_cpus	Kubernetes millicores	1000m	Number of CPU cores to be reserved for Kubelet
	native_cpu_manager_reserved_cpus	Comma-separated list of integers or integer ranges	0,1,2	Explicit list of the CPUs reserved from pods scheduling. ***Note:*** Supported only with kube_version 1.17 and newer, overrides 2 previous options.
Topology Manager (Kubernetes Built-in)[24]				
topology_manager_enabled		Boolean	true/false	Enables Kubernetes built-in Topology Manager
	topology_manager_policy	String, options: none, best-effort, restricted, single-numa-node	best-effort	Topology Manager policy
Intel SR-IOV Network Device Plugin				
sriov_network_operator_enabled		Boolean	true/false	Enables SR-IOV Network Operator
	sriov_network_operator_namespace	String	sriov-network-operator	Kubernetes namespace used to deploy SR-IOV network operator

[23] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

[24] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
sriov_net_dp_enabled		Boolean	true/false	Enables SR-IOV network device plugin
	sriov_net_dp_namespace	String	kube-system	Kubernetes namespace used to deploy SR-IOV network device plugin
	sriov_net_dp_build_image_locally	Boolean	true/false	Build and store image locally or use one from public external registry
	sriovdp_config_data	Multi-line string in JSON format	Two resource pools for kernel stack and DPDK-based networking respectively	SR-IOV network device plugin configuration. For more information on supported configurations, refer to https://github.com/intel/sriov-network-device-plugin#configurations
Intel Device Plugins for Kubernetes				
Intel_dp_namespace		String	kube-system	Kubernetes namespace used to deploy Intel device plugin operator
qat_dp_enabled		Boolean	true/false	Enables Intel QAT device plugin
	qat_dp_namespace	String	kube-system	Namespace used for Intel QAT device plugin
sgx_dp_enabled		Boolean	true/false	Enables Intel SGX device plugin
	sgx_dp_build_image_locally	Boolean	true/false	Build and store image locally or use one from public external registry
	sgx_aesmd_namespace	String	kube-system	Kubernetes namespace used to deploy SGX device plugin
	sgx_dp_provision_limit	Integer	20	
	sgx_dp_enclave_limit	Integer	20	
gpu_dp_enabled		Boolean	true	Enables Intel GPU device plugin
	gpu_dp_namespace	String	kube-system	Namespace used for Intel GPU device plugin
Intel Key Management Reference Application				
kmra_enabled		Boolean	true/false	Enables Intel Key Management Reference Application
	kmra_pccs_api_key	String	"ffffff……"	API Key obtained from Intel's Provisioning Certificate Service
	kmra_deploy_demo_workload	Boolean	true/false	Enable to deploy a KMRA demo workload (NGINX Server)
Service Mesh				
Istio_enabled		Boolean	true/false	Enables Istio service mesh for Kubernetes
Intel Telemetry Aware Scheduling				
tas_enabled		Boolean	true/false	Enables Intel Telemetry Aware Scheduling
	tas_namespace	String	monitoring	Kubernetes namespace used for TAS deployment
	tas_enable_demo_policy	Boolean	false	Creates demo TAS policy
Telemetry Configuration				
collectd_scrap_interval		Integer	30	Duration to gather metrics using collectd
telegraf_scrap_interval		Integer	30	Duration to gather metrics using Telegraf
Example Network Attachment Definitions (Ready to Use Examples of Custom CNI Plugin Configuration)				
example_net_attach_defs.		List of dictionaries	[ ]	Example network attached definition objects to create
	userspace_ovs_dpdk	Boolean	true/false	Example net-attach-def for Userspace CNI with OVS-DPDK
	userspace_vpp	Boolean	true/false	Example net-attach-def for Userspace CNI with VPP

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
	sriov_net_dp	Boolean	true/false	Example net-attach-def for SR-IOV Net DP and SR-IOV CNI

A.4 Configuration Dictionary - Host Variables

All of the variables are important but pay special attention to the variables in **bold** as they almost always need to be updated to match the target environment.

Table 26. Configuration Dictionary – Host Variables

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
SR-IOV and Network Devices Configuration				
iommu_enabled		Boolean	true/false	Sets up SR-IOV related kernel parameters and enables further SR-IOV configuration
dataplane_interfaces		**List of dictionaries**	**n/a**	**SR-IOV related NIC configuration using per-port approach**
	dataplane_interfaces[*].name	**String**	**enp24s0f0, enp24s0f1**	**Name of the interface representing PF port**
	dataplane_interfaces[*].bus_info	**String (PCI address)**	**18:00.0, 18:00.1**	**PCI address of the PF port**
	dataplane_interfaces[*].pf_driver	**String**	**ice**	**PF driver, "i40e", "ice"**
	dataplane_interfaces[*].sriov_numvfs	**Integer**	**6, 4**	**Number of VFs to be created, associated with the PF**
	dataplane_interfaces[*].default_vf_driver	**String, options: "i40evf", "iavf", "vfio-pci", "igb_uio"**	**vfio-pci for DPDK, iavf for kernel network stack**	**Default driver module name that the VFs are bound to**
	dataplane_interfaces[*].sriov_vfs[*]	**List of dictionaries**	**n/a**	**List of vfs to create with specific driver (non-default)**
	dataplane_interfaces[*].ddp_profile	**String, optional**	**gtp.pkgo**	**Name of the DDP package to be loaded onto the Network Adapter. Note: Use only for the port 0 of the Network Adapter (PCI address ending with :00.0)**
update_nic_drivers		Boolean	true/false	Set to 'true' to update Linux kernel drivers for Intel Network Adapters
update_nic_firmware		Boolean	true/false	Set 'true' to update Network Adapter firmware
	firmware_update_nics	List of strings	[enp24s0f0, enp24s0f1]	Additional list of Network Adapter interfaces that the FW update is executed on. Note: FW update is also executed on all Network Adapters listed in "dataplane_interfaces[*].name"
install_ddp_packages		Boolean	true/false	Install Intel X700 and X800 series Network Adapters DDP packages. Required if DDP packages configured in dataplane_interfaces.
install_dpdk		Boolean	true/false	DPDK installation is required for sriov_cni_enabled:true
	dpdk_version	String	21.08	DPDK version to install
	dpdk_local_patches_dir	String	Empty	Path to user-supplied patches to apply against the specified version of DPDK
SR-IOV and Bond CNI Plugins				
sriov_cni_enabled		Boolean	true/false	Installs SR-IOV CNI plugin binary on the node
bond_cni_enabled		Boolean	true/false	Installs Bond CNI plugin binary on the node
Userspace Networking Plugins and Accelerated Virtual Switches				

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
userspace_cni_enabled		Boolean	true/false	Installs userspace CNI plugin binary on the node
ovs_dpdk_enabled		Boolean	true/false	Installs OVS-DPDK on the node
	ovs_dpdk_lcore_mask	Hex integer	0x1	CPU mask for OVS-DPDK PMD threads
	ovs_dpdk_socket_mem	Integer or comma-separated list of integers	256,0	Amount of memory per NUMA node allocated to OVS-DPDK PMD threads
vpp_enabled		Boolean	true/false	Installs FD.io VPP (CentOS 7 and Ubuntu 18.04 only)
Hugepages/Memory Configuration				
hugepages_enabled		Boolean	true/false	Enables hugepages support
	default_hugepage_size	String, options: 2M, 1G	1G	Default hugepages size
	number_of_hugepages	Integer	4	Sets how many hugepages should be created
CPU Configuration				
isolcpus_enabled		Boolean	true/false	Enables CPU cores isolation from Linux scheduler
	isolcpus	Comma-separated list of CPU cores/ranges	4-11	CPU cores isolated from Linux scheduler, if CMK is enabled it's a good practice to match it with total number of shared and exclusive cores
intel_pstate		String	hwp_only	Enables Intel P-state scaling driver
	turbo_boost_enabled	Boolean	true/false	Enables Turbo Boost for P-state attribute
sst_pp_configuration_enabled		Boolean	true/false	Enables Intel SST Performance Profiles for flexible configuration of SST-BF, SST-CP, and SST-TF
	sst_pp_config_list	List of dictionaries	sst_bf: enable/disable sst_cp: enable/disable sst_tf: enable/disable	Enables configuration of SST features through SST-PP
	online_cpus_range	String	auto	Specifies automatic configuration of online CPUs versus manual configuration of each SST feature
sst_bf_configuration_enabled		Boolean	true/false	Enables Intel SST Base Frequency technology. Support of SST-BF requires 'intel_pstate' to be 'enabled'
	clx_sst_bf_mode	Character, options: s, m, r	s	Configure SST-BF mode for 2nd Generation Intel® Xeon® [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)
	icx_sst_bf_enabled	Boolean	true/false	Enables Intel SST Base Frequency technology. 3rd Generation Intel® Xeon® support of SST-BF requires 'intel_pstate' to be 'enabled'.
	icx_sst_bf_with_core_priority	Boolean	true/false	Prioritize (SST-CP) power flow to high frequency cores
sst_cp_configuration_enabled		Boolean	true/false	Enables Intel SST Core Power technology on 3rd Generation Intel® Xeon®. SST-CP overrides any 'SST-BF configuration'.
	sst_cp_priority_type	Integer	1	0 – proportional 1 - ordered

COMPONENT	COMPONENT PARAMETER	TYPE	VALUE	DESCRIPTION/COMMENT
	sst_cp_clos_groups	List of dictionaries	[ ]	Allows for configuration of up to 4 CLOS groups including id, frequency_weight, min_MHz, max_MHz
	sst_cp_cpu_clos	List of dictionaries	[ ]	Allows for definition of CPU cores per close group
sst_tf_configuration_enabled		Boolean	true/false	Enables Intel SST Turbo Frequency
Miscellaneous				
dns_disable_stub_listener	dns_disable_stub_listener	Boolean	true/false	(Ubuntu only) Disables DNS stub listener from the systemd-resolved service, which is known to cause problems with DNS and Docker containers on Ubuntu
install_real_time_package	install_real_time_package	Boolean	true/false	(CentOS 7 only) Installs real-time Linux kernel packages.
QAT Configuration				
update_qat_drivers		Boolean	true/false	Install QAT drivers and services
qat_devices		List of dictionaries	[ ]	SR-IOV related QAT configuration using per-port approach
	qat_devices[*].qat_dev	String	Crypto01, Crypto02, Crypto03	Name of the interface representing PF port
	qat_devices[*].qat_id	String (PCI address)	0000:ab:00.0, 0000:xy:00.0, 0000:yz:00.0	PCI address of the PF port
	qat_devices[*].module_type	String	qat_c62x	QAT hardware identifier, qat_c62x, qat_dh895xcc, qat_c3xxx, etc...
	qat_devices[*].pci_type	String	c6xx	PF driver, "c6xx", "c3xx", “d15xx”, etc...
	qat_devices[*].qat_sriov_numvfs	Integer	10	Number of VFs to be created per QAT device physical function
openssl_install		Boolean	true/false	Install OpenSSL for use with QAT engine

Appendix B
BMRA Basic
Configuration Profile Setup

Appendix B BMRA Basic Configuration Profile Setup

This appendix contains a step-by-step description of how to set up your BMRA Basic Configuration Profile Flavor.

To use the BMRA Basic Configuration Profile, perform the following steps:
1. Choose your hardware, set it up, and configure the BIOS. Refer to B.1 for details.
   You also need to build your Kubernetes cluster. Figure 1 is an example.
2. Download the Ansible playbook for your Configuration Profile. Refer to B.2 for details.
3. Set up the optional Ansible parameters using the information in the Configuration Profile tables. Refer to B.3 for details.
4. Deploy the platform. Refer to B.4 for details.
5. Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Be aware of the definitions of terminology used in tables in this appendix.

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

B.1 Step 1 - Set Up Basic Configuration Profile Hardware

This section describes the hardware BOM and the BIOS configuration recommendation for using the BMRA Basic Configuration Profile Flavor.

The tables in this section list the Hardware BOM for the Basic Configuration Profile, including Control Node, Worker Node Base, and Worker Node Plus. We recommend that you set up at least one control node and one worker node.

Table 27. Hardware Setup for Basic Configuration Profile – 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

NODE OPTIONS	2ND GENERATION INTEL XEON SCALABLE PROCESSOR	3RD GENERATION INTEL XEON SCALABLE PROCESSOR
Control Node Options	Controller_2ndGen_1	Controller_3rdGen_1
Worker Node Options	Worker_2ndGen_Base_1	Worker_3rdGen_Base_1

B.2 Step 2 - Download Basic Configuration Profile Ansible Playbook

This section contains step-by-step details for downloading the Basic Configuration Profile Ansible playbook. It also provides an overview of the Ansible playbook and lists the software that is automatically installed when the playbook is deployed.

Download the Basic Configuration Profile Ansible playbook using the following steps:
1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks using the tag from Table 22. For example:
   ```
   git checkout v21.09
   ```
3. Export the environmental variable for Kubernetes **Basic** Configuration Profile deployment:
   ```
   export PROFILE=basic
   ```
4. Install requirements for render.py script:
   ```
   pip3 install -r profiles/requirements.txt
   ```
5. Generate example profiles and copy the example configuration files to the project root dir:
   ```
   make bmra-profiles profile=$PROFILE
   ```

B.2.1 Basic Configuration Profile Ansible Playbook Overview

The Ansible playbook for the Basic Configuration Profile allows you to provision a production-ready Kubernetes cluster. Every capability included in the Basic Configuration Profile playbook can be disabled or enabled. Refer to the diagram and group and host variables tables below to see which Ansible roles are included and executed by default.

The diagram shows the architecture of the Ansible playbooks and roles that are included in the Basic Configuration Profile.

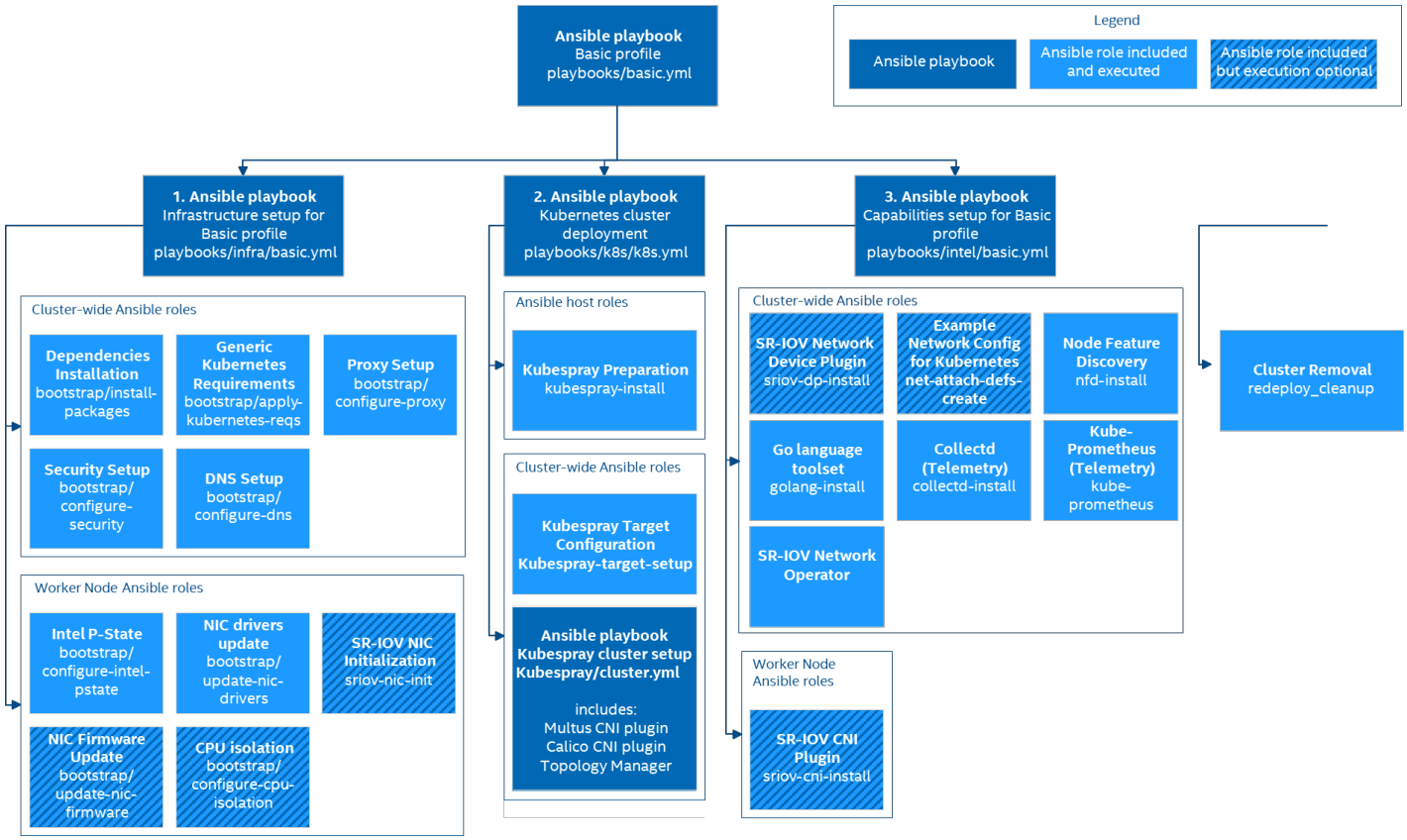

Figure 13. Basic Configuration Profile Ansible Playbook

B.3 Step 3 - Set Up Basic Configuration Profile

Review the optional Ansible group and host variables in this section and select options that match your desired configuration.

1. Update the `inventory.ini` file with your environment details as described in Section 3.3.3.
2. Create host_vars files for all worker nodes specified in the inventory. For example, if you have worker1, worker2, and worker3 in the kube-node group, execute:

```
mv host_vars/node1.yml host_vars/worker1.yml
cp host_vars/worker1.yml host_vars/worker2.yml
cp host_vars/worker1.yml host_vars/worker3.yml
```

3. Update group and host variables to match your desired configuration. Refer to the tables in B.3.1 and B.3.2.

   Note: Pay special attention to the variables in **bold** as these almost always need to be updated individually to match your environment details. Make sure that <worker_node_name>.yml files have been created for all worker nodes specified in your inventory file.

```
vim group_vars/all.yml
vim host_vars/<worker_node_name>.yml
```

The complete set of configuration variables for the Basic Configuration Profile along with their default values can be found in examples/basic directory.

Variables are grouped into two main categories:

1. Group variables – apply to both control and worker nodes and have cluster-wide impact.
2. Host variables – their scope is limited to a single worker node.

The tables below are a summary of group and host variables. For lists showing all configurable properties, see Section A.3 and Section A.4. All of the variables are important but pay special attention to variables in **bold** as they almost always need to be updated to match the target environment.

B.3.1 Basic Configuration Profile Group Variables

Table 28. Basic Configuration Profile – Group Variables

COMPONENT	VALUE	
Kubernetes	true	For the list of all configurable properties, see Section A.3
nfd_enabled	true	
topology_manager_enabled	true	
sriov_network_operator_enabled	false	
sriov_net_dp_enabled	false	
example_net_attach_defs, sriov_net_dp	false	

B.3.2 Basic Configuration Profile Host Variables[25]

Table 29. Basic Configuration Profile – Host Variables

COMPONENT	VALUE	
iommu_enabled	false	For the list of all configurable properties, see Section A.4
sriov_cni_enabled	false	
isolcpus_enabled	false	
dns_disable_stub_listener	true	

B.4 Step 4 - Deploy Basic Configuration Profile Platform

Note: You must download the Configuration Profile playbook as described in B.2 and set it up as described in B.3 before you complete this step.

In order to deploy the Basic Configuration Profile playbook, change the working directory to where you have cloned or unarchived the BMRA Ansible Playbook source code (as described in Section 3.3.2) and execute the command below:

```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

B.5 Step 5 - Validate Basic Configuration Profile

Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

[25] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Appendix C
BMRA Full
Configuration Profile Setup

Appendix C BMRA Full Configuration Profile Setup

This appendix contains a step-by-step description of how to set up your BMRA Full Configuration Profile Flavor.

To use the BMRA Full Configuration Profile, perform the following steps:
1. Choose your hardware, set it up, and configure the BIOS. Refer to C.1 for details.
   You also need to build your Kubernetes cluster. Figure 1 is an example.
2. Download the Ansible playbook for your Configuration Profile. Refer to C.2 for details.
3. Configure the optional Ansible parameters using the information in the Configuration Profile tables. Refer to C.3 for details.
4. Deploy the platform. Refer to C.4 for details.
5. Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Be aware of the definitions of terminology used in tables in this appendix.

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value.)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

C.1 Step 1 - Set Up Full Configuration Profile Hardware

This section describes the hardware BOM and the BIOS configuration recommendation for using the BMRA Full Configuration Profile Flavor.

The tables in this section list the Hardware BOM for the Full Configuration Profile, including Control Node, Worker Node Base, and Worker Node Plus. We recommend that you set up at least three control nodes and two worker nodes.

Table 30. Hardware Setup for Full Configuration Profile – 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

NODE OPTIONS	2ND GENERATION INTEL XEON SCALABLE PROCESSOR	3RD GENERATION INTEL XEON SCALABLE PROCESSOR
Control Node Options	Controller_2ndGen_3	Controller_3rdGen_3
Worker Node Options	Worker_2ndGen_Plus_1	Worker_3rdGen_Plus_1

C.2 Step 2 - Download Full Configuration Profile Ansible Playbook

This section contains step-by-step details for downloading the Full Configuration Profile Ansible playbook. It also provides an overview of the Ansible playbook and lists the software that is automatically installed when the playbook is deployed.

Download the Full Configuration Profile Ansible playbook using the following steps:
1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks using the tag from Table 22. For example:
   ```
   git checkout v21.09
   ```
3. Export the environmental variable for Kubernetes **Full** Configuration Profile deployment:
   ```
   export PROFILE=full_nfv
   ```
4. Install requirements for render.py script:
   ```
   pip3 install -r profiles/requirements.txt
   ```
5. Generate example profiles and copy the example configuration files to the project root dir:
   ```
   make bmra-profiles profile=$PROFILE
   ```

C.2.1 Full Configuration Profile Ansible Playbook Overview

The Ansible playbook for the Full Configuration Profile allows you to provision a production-ready Kubernetes. It also applies any additional requirements, such as host OS configuration or Network Adapter drivers and firmware updates. Full Configuration Profile playbook includes all features available through BMRA Ansible Playbook and provides one of the highest degrees of configurability. Every capability included in the Full Configuration Profile playbook can be disabled or enabled. Refer to the diagram and group and host variables tables below to see which Ansible roles are included and executed by default.

The diagram shows the architecture of the Ansible playbooks and roles that are included in the Full Configuration Profile.

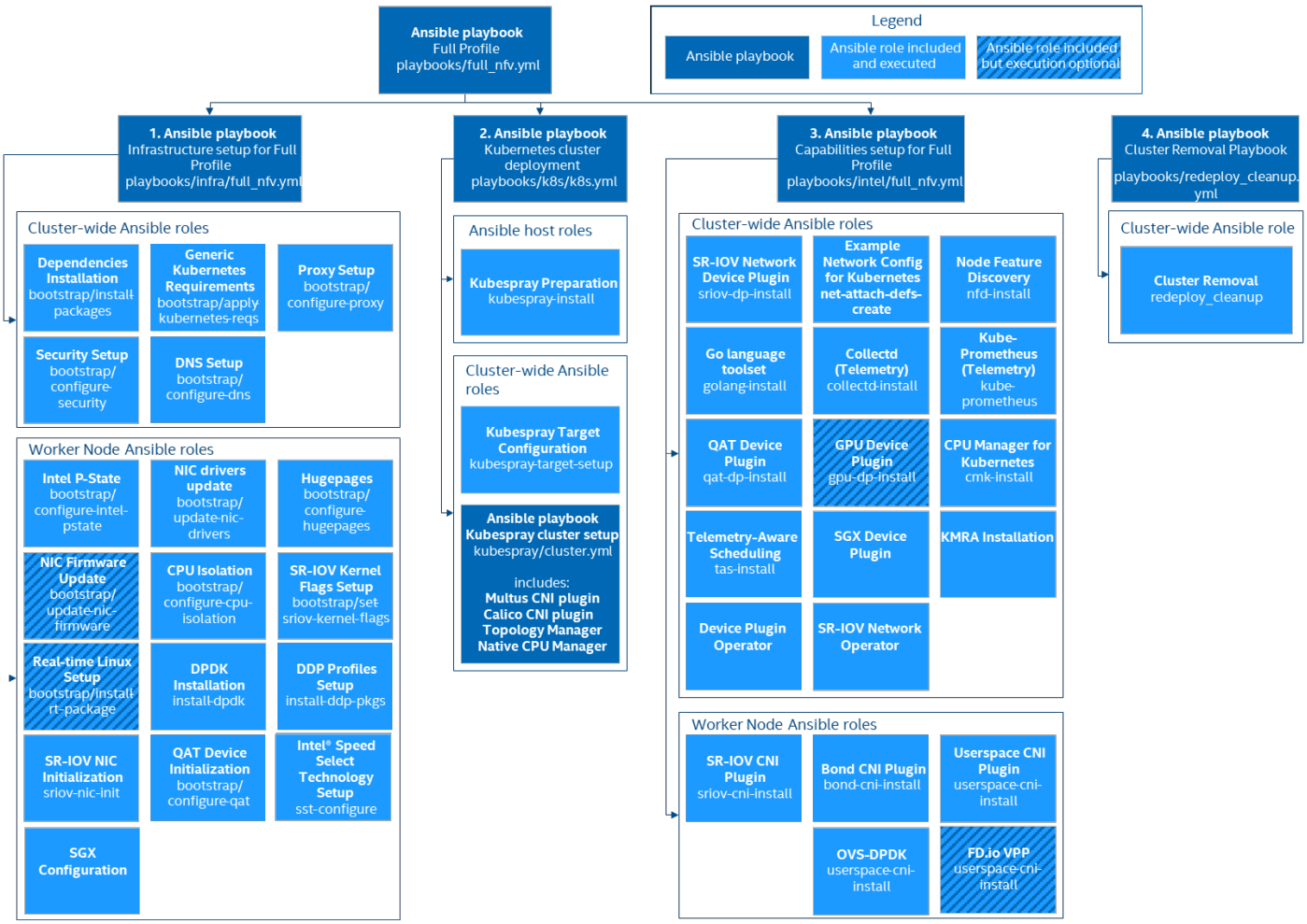

Figure 14. Full Configuration Profile Ansible Playbook

C.3 Step 3 - Set Up Full Configuration Profile

Review the optional Ansible group and host variables in this section and select options that match your desired configuration.

1. Update the `inventory.ini` file with your environment details as described in Section 3.3.3.
2. Create host_vars files for all worker nodes specified in the inventory. For example, if you have worker1, worker2, and worker3 in the kube-node group, execute:

```
mv host_vars/node1.yml host_vars/worker1.yml
cp host_vars/worker1.yml host_vars/worker2.yml
cp host_vars/worker1.yml host_vars/worker3.yml
```

3. Update group and host variables to match your desired configuration. Refer to the tables in Section C.3.1 and Section C.3.2.

   Note: Pay special attention to the variables in **bold** as these almost always need to be updated individually to match your environment details. Make sure that <worker_node_name>.yml files have been created for all worker nodes specified in your inventory file.

```
vim group_vars/all.yml
vim host_vars/<worker_node_name>.yml
```

The complete set of configuration variables for the Full Configuration Profile along with their default values can be found in the examples/full_nfv directory.

Variables are grouped into two main categories:
1. Group variables – they apply to both control and worker nodes and have cluster-wide impact.
2. Host variables – their scope is limited to a single worker node.

The tables below are a summary of group and host variables. For lists showing all configurable properties, see Section A.3 and Section A.4. All of the variables are important but pay special attention to variables in **bold** as they almost always need to be updated to match the target environment.

C.3.1 Full Configuration Profile Group Variables

Table 31. Full Configuration Profile – Group Variables

COMPONENT	VALUE	
Kubernetes	true	For the list of all configurable properties, see Section A.3
nfd_enabled	true	
cmk_enabled	true	
native_cpu_manager_enabled	false	
topology_manager_enabled	true	
sriov_network_operator_enabled	true	
sriov_net_dp_enabled	false	
sgx_dp_enabled	true	
gpu_dp_enabled	true	
qat_dp_enabled	true	
openssl_enabled	true	
kmra_enabled	true	
istio_enabled	true	
tas_enabled	true	
sst_pp_configuration_enabled	true	
example_net_attach_defs. userspace_ovs_dpdk	true	

C.3.2 Full Configuration Profile Host Variables[26]

Table 32. Full Configuration Profile – Host Variables

COMPONENT	VALUE	
iommu_enabled	true	For the list of all configurable properties, see Section A.4
sriov_cni_enabled	true	
bond_cni_enabled	true	
userspace_cni_enabled	true	
hugepages_enabled	true	
isolcpus_enabled	true	
dns_disable_stub_listener	true	
install_dpdk	true	
install_ddp_packages	true	
install_real_time_package	false	
qat_devices	**[ ]**	

[26] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

C.4 Step 4 - Deploy Full Configuration Profile Platform

Note: You must download the Configuration Profile playbook as described in C.2 and configure it as described in C.3 before you complete this step.

In order to deploy the Full Configuration Profile playbook, change the working directory to where you have cloned or unarchived the BMRA Ansible Playbook source code (as described in Section 3.3.2) and execute the command below:

```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

C.5 Step 5 - Validate Full Configuration Profile

Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Appendix D
BMRA On-Premises Edge Configuration Profile Setup

Appendix D BMRA On-Premises Edge Configuration Profile Setup

This appendix contains a step-by-step description of how to set up your BMRA On-Premises Edge Configuration Profile Flavor.

To use the BMRA On-Premises Edge Configuration Profile, perform the following steps:
1. Choose your hardware, set it up, and configure the BIOS. Refer to D.1 for details.
   You also need to build your Kubernetes cluster. Figure 1 is an example.
2. Download the Ansible playbook for your Configuration Profile. Refer to D.2 for details.
3. Configure the optional Ansible parameters using the information in the Configuration Profile tables. Refer to D.3 for details.
4. Deploy the platform. Refer to D.4 for details.
5. Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Be aware of the definitions of terminology used in tables in this appendix.

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value.)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

D.1 Step 1 - Set Up On-Premises Edge Configuration Profile Hardware

The tables in this section list the Hardware BOM for the On-Premises Edge Configuration Profile, including Control Node, Worker Node Base, and Worker Node Plus.

We recommend that you set up at least one control node and one worker node.

Table 33. Hardware Setup for On-Premises Edge Configuration Profile – 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

NODE OPTIONS	2ND GENERATION INTEL XEON SCALABLE PROCESSOR	3RD GENERATION INTEL XEON SCALABLE PROCESSOR
Control Node Options	Controller_2ndGen_1	Controller_3rdGen_1
Worker Node Options	Worker_2ndGen_Base_2 or Worker_2ndGen_Plus_1	Worker_3rdGen_Base_2 or Worker_3rdGen_Plus_1

D.2 Step 2 - Download On-Premises Edge Configuration Profile Ansible Playbook

This section contains step-by-step details for downloading the On-Premises Edge Configuration Profile Ansible playbook. It also provides an overview of the Ansible playbook and lists the software that is automatically installed when the playbook is deployed.

Download the On-Premises Edge Configuration Profile Ansible playbook using the following steps:
1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks using the tag from Table 22. For example:
   ```
   git checkout v21.09
   ```
3. Export the environmental variable for Kubernetes **On-Premises Edge** Configuration Profile deployment:
   ```
   export PROFILE=on_prem
   ```
4. Install requirements for render.py script:
   ```
   pip3 install -r profiles/requirements.txt
   ```
5. Generate example profiles and copy the example configuration files to the project root dir:
   ```
   make bmra-profiles profile=$PROFILE
   ```

D.2.1 On-Premises Edge Configuration Profile Ansible Playbook Overview

The Ansible playbook for the On-Premises Edge Configuration Profile allows you to provision a production-ready Kubernetes cluster. It also applies any additional requirements, such as host OS configuration or Network Adapter drivers and firmware updates. Every capability included in the On-Premises Edge Configuration Profile playbook can be disabled or enabled. Refer to the diagram and group and host variables tables below to see which Ansible roles are included and executed by default.

The diagram shows the architecture of the Ansible playbooks and roles that are included in the On-Premises Edge Configuration Profile.

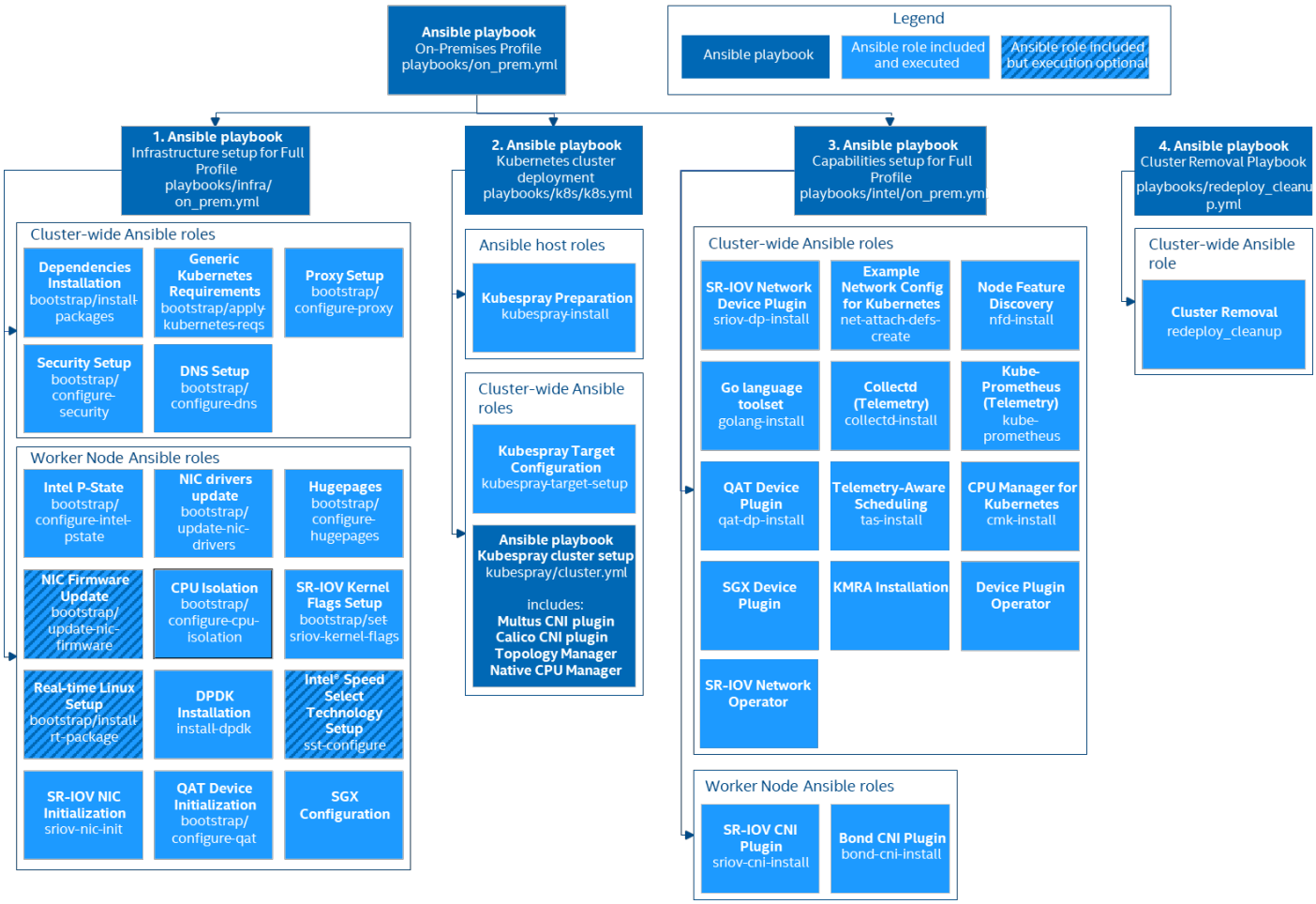

Figure 15. On-Premises Edge Configuration Profile Ansible Playbook

D.3 Step 3 - Set Up On-Premises Edge Configuration Profile

Review the optional Ansible group and host variables in this section and select options that match your desired configuration.

1. Update the `inventory.ini` file with your environment details as described in Section 3.3.3.
2. Create host_vars files for all worker nodes specified in the inventory. For example, if you have worker1, worker2 and worker3 in the kube-node group, execute:

```
mv host_vars/node1.yml host_vars/worker1.yml
cp host_vars/worker1.yml host_vars/worker2.yml
cp host_vars/worker1.yml host_vars/worker3.yml
```

3. Update group and host variables to match your desired configuration. Refer to the tables in Section D.3.1 and Section D.3.2.

   Note: Pay special attention to the variables in **bold** as these almost always need to be updated individually to match your environment details. Make sure that <worker_node_name>.yml files have been created for all worker nodes specified in your inventory file.

```
vim group_vars/all.yml
vim host_vars/<worker_node_name>.yml
```

The complete set of configuration variables for the On-Premises Edge Configuration Profile along with their default values can be found in the examples/on_prem directory.

Variables are grouped into two main categories:

1. Group variables – they apply to both control and worker nodes and have cluster-wide impact.
2. Host variables – their scope is limited to a single worker node.

The tables below are a summary of group and host variables. For lists showing all configurable properties, see Section A.3 and Section A.4. All of the variables are important but pay special attention to variables in **bold** as they almost always need to be updated to match the target environment.

D.3.1 On-Premises Edge Configuration Profile Group Variables

Table 34. On-Premises Edge Configuration Profile – Group Variables

COMPONENT	VALUE	
Kubernetes	true	For the list of all configurable properties, see Section A.3
nfd_enabled	true	
cmk_enabled	true	
native_cpu_manager_enabled	false	
topology_manager_enabled	true	
sriov_network_operator_enabled	true	
sriov_net_dp_enabled	false	
sgx_dp_enabled	true	
gpu_dp_enabled	false	
qat_dp_enabled	true	
openssl_enabled	true	
kmra_enabled	true	
istio_enabled	true	
tas_enabled	true	
sst_pp_configuration_enabled	false	
example_net_attach_defs. userspace_ovs_dpdk	false	

D.3.2 On-Premises Edge Configuration Profile Host Variables[27]

Table 35. On-Premises Edge Configuration Profile – Host Variables

COMPONENT	VALUE	
iommu_enabled	true	For the list of all configurable properties, see Section A.4
sriov_cni_enabled	false	
bond_cni_enabled	false	
hugepages_enabled	true	
isolcpus_enabled	true	
dns_disable_stub_listener	true	
install_dpdk	true	
install_real_time_package	false	
qat_devices	**[ ]**	

D.4 Step 4 - Deploy On-Premises Edge Configuration Profile Platform

Note: You must download the Configuration Profile playbook as described in D.2 and configure it as described in D.3 before you complete this step.

[27] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

In order to deploy the On-Premises Edge Configuration Profile playbook, change the working directory to where you have cloned or unarchived the BMRA Ansible Playbook source code (as described in Section 3.3.2) and execute the command below:

```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

D.5 Step 5 - Validate On-Premises Edge Configuration Profile

Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Appendix E
BMRA Remote Central Office-Forwarding Configuration Profile Setup

Appendix E BMRA Remote CO-Forwarding Configuration Profile Setup

This appendix contains a step-by-step description of how to set up your BMRA Remote CO-Forwarding Configuration Profile Flavor.

To use the Remote CO-Forwarding Configuration Profile, perform the following steps:
1. Choose your hardware, set it up, and configure the BIOS. Refer to E.1 for details.
   You also need to build your Kubernetes cluster. Figure 1 is an example.
2. Download the Ansible playbook for your Configuration Profile. Refer to E.2 for details.
3. Configure the optional Ansible parameters using the information in the Configuration Profile tables. Refer to E.3 for details.
4. Deploy the platform. Refer to E.4 for details.
5. Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Be aware of the definitions of terminology used in tables in this appendix.

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value.)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

E.1 Step 1 - Set Up Remote CO-Forwarding Configuration Profile Hardware

The tables in this section list the Hardware BOM for the Remote CO-Forwarding Configuration Profile, including Control Node, Worker Node Base, and Worker Node Plus.

We recommend that you set up at least one control node and one worker node.

Table 36. Hardware Setup for Remote CO-Forwarding Configuration Profile – 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

NODE OPTIONS	2ND GENERATION INTEL XEON SCALABLE PROCESSOR	3RD GENERATION INTEL XEON SCALABLE PROCESSOR
Control Node Options	Controller_2ndGen_2	Controller_3rdGen_2
Worker Node Options	Worker_2ndGen_Base_3 or Worker_2ndGen_Plus_2	Worker_3rdGen_Base_3 or Worker_3rdGen_Plus_2

E.2 Step 2 - Download Remote CO-Forwarding Configuration Profile Ansible Playbook

This section contains step-by-step details for downloading the Remote CO-Forwarding Configuration Profile Ansible playbook. It also provides an overview of the Ansible playbook and lists the software that is automatically installed when the playbook is deployed.

Download the Remote CO-Forwarding Configuration Profile Ansible playbook using the following steps:
1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks using the tag from Table 22. For example:
   ```
   git checkout v21.09
   ```
3. Export the environmental variable for Kubernetes **Remote CO-Forwarding** Configuration Profile deployment:
   ```
   export PROFILE=remote_fp
   ```
4. Install requirements for render.py script:
   ```
   pip3 install -r profiles/requirements.txt
   ```

5. Generate example profiles and copy the example configuration files to the project root dir:

```
make bmra-profiles profile=$PROFILE
```

E.2.1 Remote CO-Forwarding Configuration Profile Ansible Playbook Overview

The Ansible playbook for the Remote CO-Forwarding Configuration Profile allows you to provision a production-ready Kubernetes cluster. It also applies any additional requirements, such as host OS configuration or Network Adapter drivers and firmware updates. Every capability included in the Remote CO-Forwarding Configuration Profile playbook can be disabled or enabled. Refer to the diagram and group and host variables tables below to see which Ansible roles are included and executed by default.

The diagram shows the architecture of the Ansible playbooks and roles that are included in the Remote CO-Forwarding Configuration Profile.

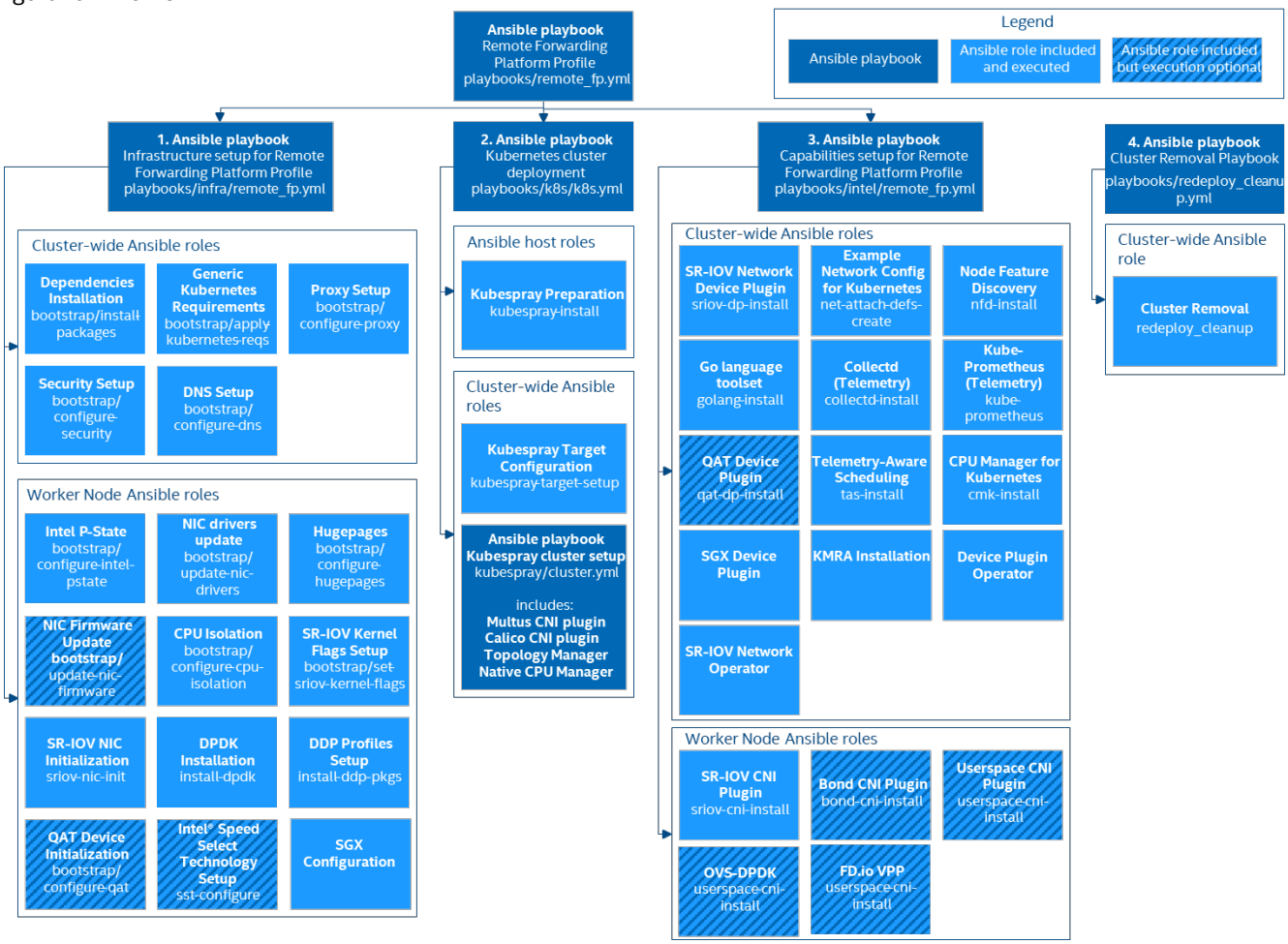

Figure 16. Remote CO-Forwarding Configuration Profile Ansible Playbook

E.3 Step 3 - Set Up Remote CO-Forwarding Configuration Profile

Review the optional Ansible group and host variables in this section and select options that match your desired configuration.

1. Update the `inventory.ini` file with your environment details as described in Section 3.3.3.
2. Create host_vars files for all worker nodes specified in the inventory. For example, if you have worker1, worker2, and worker3 in the kube-node group, execute:

```
mv host_vars/node1.yml host_vars/worker1.yml
cp host_vars/worker1.yml host_vars/worker2.yml
cp host_vars/worker1.yml host_vars/worker3.yml
```

3. Update group and host variables to match your desired configuration. Refer to the tables in Section E.3.1 and Section E.3.2.

   Note: Pay special attention to the variables in **bold** as these almost always need to be updated individually to match your environment details. Make sure that <worker_node_name>.yml files have been created for all worker nodes specified in your inventory file.

```
vim group_vars/all.yml
vim host_vars/<worker_node_name>.yml
```

The complete set of configuration variables for the Remote CO-Forwarding Configuration Profile along with their default values can be found in examples/remote_fp directory.

Variables are grouped into two main categories:
1. Group variables – they apply to both control and worker nodes and have cluster-wide impact.
2. Host variables – their scope is limited to a single worker node.

The tables below are a summary of group and host variables. For lists showing all configurable properties, see Section A.3 and Section A.4. All of the variables are important but pay special attention to variables in **bold** as they almost always need to be updated to match the target environment.

E.3.1 Remote CO-Forwarding Configuration Profile Group Variables

Table 37. Remote CO-Forwarding Configuration Profile – Group Variables

COMPONENT	VALUE	
Kubernetes	true	For the list of all configurable properties, see Section A.3
nfd_enabled	true	
cmk_enabled	true	
native_cpu_manager_enabled	false	
topology_manager_enabled	true	
sriov_network_operator_enabled	true	
sriov_net_dp_enabled	false	
sgx_dp_enabled	true	
gpu_dp_enabled	false	
qat_dp_enabled	false	
openssl_enabled	true	
kmra_enabled	true	
istio_enabled	true	
tas_enabled	true	
sst_cp_configuration_enabled	false	
example_net_attach_defs. userspace_ovs_dpdk	false	

E.3.2 Remote CO-Forwarding Configuration Profile Host Variables[28]

Table 38. Remote CO-Forwarding Configuration Profile – Host Variables

COMPONENT	VALUE	
iommu_enabled	true	For the list of all configurable properties, see Section A.4
sriov_cni_enabled	false	
bond_cni_enabled	false	
userspace_cni_enabled	false	
hugepages_enabled	true	
isolcpus_enabled	true	
dns_disable_stub_listener	true	
install_dpdk	true	
install_ddp_packages	true	
install_real_time_package	false	
qat_devices	**[ ]**	

[28] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

E.4 Step 4 - Deploy Remote CO-Forwarding Configuration Profile Platform

Note: You must download the Configuration Profile playbook as described in E.2 and configure it as described in E.3 before you complete this step.

In order to deploy the Remote CO-Forwarding Configuration Profile playbook, change the working directory to where you have cloned or unarchived the BMRA Ansible Playbook source code (as described in Section 3.3.2) and execute the command below:

```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

E.5 Step 5 - Validate Remote-CO Forwarding Configuration Profile

Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Appendix F
BMRA Regional Data Center Configuration Profile Setup

Appendix F BMRA Regional Data Center Configuration Profile Setup

This appendix contains a step-by-step description of how to set up your BMRA Regional Data Center Configuration Profile Flavor.

To use the Regional Data Center Configuration Profile, perform the following steps:
1. Choose your hardware, set it up, and configure the BIOS. Refer to F.1 for details.
   You also need to build your Kubernetes cluster. Figure 1 is an example.
2. Download the Ansible playbook for your Configuration Profile. Refer to F.2 for details.
3. Configure the optional Ansible parameters using the information in the Configuration Profile tables. Refer to F.3 for details.
4. Deploy the platform. Refer to F.4 for details.
5. Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

Be aware of the definitions of terminology used in tables in this appendix.

TERM	DESCRIPTION
Hardware Taxonomy	
ENABLED	Setting must be enabled in the BIOS (configured as Enabled, Yes, True, or similar value.)
DISABLED	Setting must be disabled in the BIOS (configured as Disabled, No, False, or any other value with this meaning.)
OPTIONAL	Setting can be either disabled or enabled, depending on user's workload. Setting does not affect the Configuration Profile or platform deployment.
Software Taxonomy	
TRUE	Feature is included and enabled by default.
FALSE	Feature is included but disabled by default - can be enabled and configured by user.
N/A	Feature is not included and cannot be enabled or configured.

F.1 Step 1 - Set Up Regional Data Center Configuration Profile Hardware

The tables in this section list the Hardware BOM for the Regional Data Center Configuration Profile, including Control Node, Worker Node Base, and Worker Node Plus.

We recommend that you set up at least one control node and one worker node.

Table 39. Hardware Setup for Regional Data Center Configuration Profile – 2nd Generation and 3rd Generation Intel Xeon Scalable Processors

NODE OPTIONS	2ND GENERATION INTEL XEON SCALABLE PROCESSOR	3RD GENERATION INTEL XEON SCALABLE PROCESSOR
Control Node Options	N/A*	Controller_3rdGen_3
Worker Node Options	N/A*	Worker_3rdGen_Plus_3
*Configuration Profile only tested with 3rd Generation Intel Xeon Scalable processor		

F.2 Step 2 - Download Regional Data Center Configuration Profile Ansible Playbook

This section contains step-by-step details for downloading the Regional Data Center Configuration Profile Ansible playbook. It also provides an overview of the Ansible playbook and lists the software that is automatically installed when the playbook is deployed.

Download the Regional Data Center Configuration Profile Ansible playbook using the following steps:
1. Log in to your Ansible host (the one that you will run these Ansible playbooks from).
2. Clone the source code and change working directory:
   ```
   git clone https://github.com/intel/container-experience-kits/
   cd container-experience-kits
   ```
   Check out the latest version of the playbooks using the tag from Table 22. For example:
   ```
   git checkout v21.09
   ```
3. Export the environmental variable for Kubernetes **BMRA Regional Data Center** deployment:
   ```
   export PROFILE=regional_dc
   ```
4. Install requirements for render.py script:
   ```
   pip3 install -r profiles/requirements.txt
   ```
5. Generate example profiles and copy the example configuration files to the project root dir:
   ```
   make bmra-profiles profile=$PROFILE
   ```

F.2.1 Regional Data Center Configuration Profile Ansible Playbook Overview

The Ansible playbook for the Regional Data Center Configuration Profile allows you to provision a production-ready Kubernetes cluster. It also applies any additional requirements, such as host OS configuration or Network Adapter drivers and firmware updates. Every capability included in the Regional Data Center Configuration Profile playbook can be disabled or enabled. Refer to the diagram and group and host vars tables below to see which Ansible roles are included and executed by default.

The diagram shows the architecture of the Ansible playbooks and roles that are included in the Regional Data Center Configuration Profile.

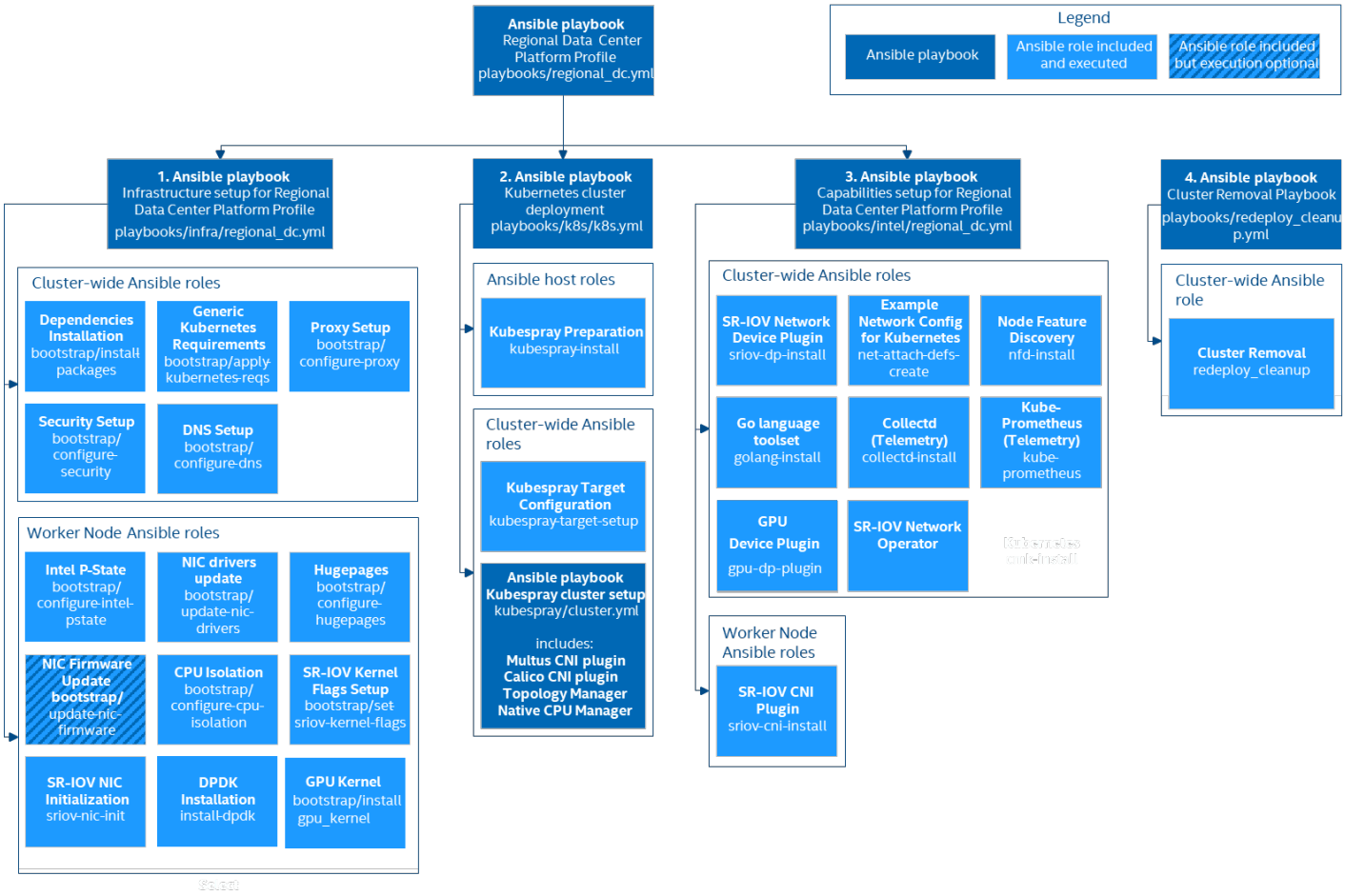

Figure 17. Regional Data Center Configuration Profile Ansible Playbook

F.3 Step 3 - Set Up Regional Data Center Configuration Profile

Review the optional Ansible group and host variables in this section and select options that match your desired configuration.

1. Update the `inventory.ini` file with your environment details as described in Section 3.3.3.
2. Create host_vars files for all worker nodes specified in the inventory. For example, if you have worker1, worker2, and worker3 in the kube-node group, execute:

```
mv host_vars/node1.yml host_vars/worker1.yml
cp host_vars/worker1.yml host_vars/worker2.yml
cp host_vars/worker1.yml host_vars/worker3.yml
```

3. Update group and host variables to match your desired configuration. Refer to the tables in Section F.3.1 and Section F.3.2.

   Note: Pay special attention to the variables in **bold** as these almost always need to be updated individually to match your environment details. Make sure that <worker_node_name>.yml files have been created for all worker nodes specified in your inventory file.

```
vim group_vars/all.yml
vim host_vars/<worker_node_name>.yml
```

The complete set of configuration variables for the Regional Data Center Configuration Profile along with their default values can be found in examples/regional_dc directory.

Variables are grouped into two main categories:
1. Group variables – they apply to both control and worker nodes and have cluster-wide impact.
2. Host variables – their scope is limited to a single worker node.

The tables below are a summary of group and host variables. For lists showing all configurable properties, see Section A.3 and Section A.4. All of the variables are important but pay special attention to variables in **bold** as they almost always need to be updated to match the target environment.

F.3.1 Regional Data Center Configuration Profile Group Variables

Table 40. Regional Data Center Configuration Profile – Group Variables

COMPONENT	VALUE	
Kubernetes	true	For the list of all configurable properties, see Section A.3
nfd_enabled	true	
native_cpu_manager_enabled	false	
topology_manager_enabled	true	
sriov_network_operator_enabled	false	
sriov_net_dp_enabled	false	
gpu_dp_enabled	true	
Istio_enabled	true	
tas_enabled	true	
example_net_attach_defs. sriov_net_dp	false	

F.3.2 Regional Data Center Configuration Profile Host Variables[29]

Table 41. Regional Data Center Configuration Profile – Host Variables

COMPONENT	VALUE	
iommu_enabled	false	For the list of all configurable properties, see Section A.4
sriov_cni_enabled	false	
hugepages_enabled	false	
isolcpus_enabled	false	
dns_disable_stub_listener	true	
install_dpdk	false	

F.4 Step 4 - Deploy Regional Data Center Configuration Profile Platform

Note: You must download the Configuration Profile playbook as described in F.2 and configure it as described in F.3 before you complete this step.

In order to deploy the Regional Data Center Configuration Profile playbook, change the working directory to where you have cloned or unarchived the BMRA Ansible Playbook source code (as described in Section 3.3.2) and execute the command below:

```
ansible-playbook -i inventory.ini playbooks/${PROFILE}.yml
```

F.5 Step 5 - Validate Regional Data Center Configuration Profile

Validate the setup of your Kubernetes cluster. Refer to the tasks in Section 7 and run the validation processes according to the hardware and software components that you have installed.

[29] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Part 4:
Appendix G
BMRA 21.09 Release Notes

Appendix G BMRA Release Notes

This appendix lists the notable changes from the previous releases, including new features, bug fixes, and known issues.[30]

G.1 BMRA 21.09 New Features

The following new features were updated or added in this release:

- Support for Istio service mesh operator, Envoy, and control plane
- Support for Telegraf telemetry collection
- Support Intel Telemetry Insight Reports
- Support for additional container runtime: CRI-O
- Updated default network plugin: Calico
- Support Intel® Speed Select Technology - Performance Profile (Intel® SST-PP)
- Support for rendering profile config files from template
- Updated Intel® Ethernet 700 and 800 Network Adapter drivers
- Updated Intel® Software Guard Extensions (Intel® SGX) Software Development Kit (SDK)
- Updated Data Plane Development Kit (DPDK) and Open vSwitch (OVS) DPDK for use of AVX-512 instruction sets
- Updated Prometheus, Grafana, and Node Exporter telemetry packages
- Updated Node Feature Discovery (NFD)
- Updated Multus container network interface (CNI)
- Updated OpenSSL toolkit
- Updated Intel® QuickAssist Technology Engine for OpenSSL (Intel® QAT Engine for OpenSSL)
- Updated Intel® Multi-Buffer Crypto for IPSec (intel-ipsec-mb)

G.2 BMRA 21.09 Bug Fixes

The following bug fixes were completed in the BMRA 21.09 release:

- Fixed inventory groups for inclusive terminology
- Fixed kubelet –cpu-cfs-quota to eliminate performance throttling
- Fixed QAT driver VF binding issue on RHEL 8.4
- Fixed inadvertent Intel SST-CP frequency throttling with proportional settings

G.3 BMRA 21.08 New Features

The following new features were updated or added in this release:

- Updated Intel® Ethernet 700 and 800 Network Adapter drivers
- Updated Intel® Ethernet 800 Dynamic Device Personalization (DDP) profiles
- Updated Intel device plugins (Intel QAT, Intel® Software Guard Extensions (Intel® SGX), Intel® Server GPU)
- Support additional operating system versions: RHEL 8.4 and Ubuntu 21.04
- Support additional container runtime: containerd
- Support Kubernetes version 1.21
- Updated Kubernetes features: Node Feature Discovery (NFD), Telemetry Aware Scheduling (TAS), and SR-IOV device plugin (DP)
- Support Kubernetes Operators for: Intel® device plugin operator (Intel SGX, Intel Server GPU) and SR-IOV network
- Support Intel® QuickAssist Technology Engine for OpenSSL (Intel® QAT Engine for OpenSSL)
- Support containerized Intel® SGX Key Management Service (KMS) including integration of Key Management Reference Application (KMRA) version 1.2.1
- Updated Collectd, Prometheus, and Grafana components
- Support for DPDK 21.05 and OVS 2.15
- Support Ansible Cluster Removal Playbook for cluster teardown and redeployment

G.4 BMRA 21.08 Bug Fixes

The following bug fixes were completed in the BMRA 21.08 release:

[30] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

- Fixed cluster recovery errors on reboot
- Fixed TAS demo policy failure
- Fixed deployment failure with Intel® Turbo Boost Technology disabled
- Fixed SGX DP pod crashes
- Fixed mismatch in available Intel QAT resources
- Fixed deployment failure with missing Intel QAT configuration
- Fixed kernel header mismatch for compiled kernel modules
- Fixed package installation dependencies
- Fixed isolcpu generation for configurations with HT disabled
- Fixed DPDK installation failures
- Fixed DNS file permission issues
- Fixed IP dependency in inventory file

G.5 BMRA 21.03 New Features

The following new features were updated or added in this release:

- Kubernetes version update to 1.19.x
- Kubernetes feature/plugin updates (NFD, TAS, SR-IOV-DP)
- CentOS 8.3 support
- RHEL 8.3 support
- CentOS 7.9 support
- 3rd Generation Intel® Xeon® Scalable processor support
- Intel® Software Guard Extensions device plugin
- Intel® SGX Key Management Services
- Intel® QuickAssist Technology Drivers and Services
- Intel® Speed Select Technology – Core Power (Intel® SST-CP)
- Intel® Server Graphics 1 card support
- Updated Intel® Ethernet 700 & 800 Network Adapter Drivers and DDP profiles
- New Regional Data Center Configuration Profile for Visual Compute Media workloads using Intel® Server Graphics 1
- Additional Collectd plugins (unixsock, network)
- Additional Grafana dashboards (cpu, disk, intel, ipmi, netlink, ovs, power, numa, hugepages, ethstats)
- Multiple DPDK version options with custom patch support
- Multiple SR-IOV driver assignments per PF

G.6 BMRA 21.03 Bug Fixes

The following bug fixes were completed in the BMRA 21.03 release:

- Fixed Intel® Network Adapter driver compilation on RHEL
- Fixed Comms DDP profiles not loading at OS boot time
- Cleaned up Ansible warnings occurring during playbook runtime
- Forced DPDK bindings for active devices defined in host_vars
- Fixed CMK installation failure across multiple worker nodes
- Fixed certificate handling causing Prometheus pod failures

G.7 BMRA 2.1 New Features

No new features were added.

G.8 BMRA 2.1 Bug Fixes

The following bug fixes were completed in the BMRA 2.1 release:

- Intel® Ice driver fails to load on Ubuntu 18.04
- Updated Intel Network Adapter drivers to resolve driver compilation issues
- Increased driver download timeouts
- Removed duplicate collectd install in remote_fp profile
- Added missing NFD role to remote_fp profile
- Fixed CMK container template to include cmk binary

- Cleaned up data plane interface and IOMMU terminology
- Defaulted data plane interface examples as empty lists
- Updated Intel Network Adapter DDP profile URL
- Updated Intel Network Adapter FW package URL
- Fixed data plane interface selection on DDP profile loading
- Updated Helm stable repo URL
- Fixed DPDK vfio-pci binding for Ubuntu 18.04 and 20.04
- Updated default DPDK version to fix compilation on CentOS/RHEL 8
- Fixed PowerTools repository names for CentOS 8

G.9 BMRA 2.0 New Features

The following new features were added in this release:

- Ubuntu 20.04 support
- CentOS 8.2 support
- Red Hat Enterprise Linux 8.2 support
- Intel® Speed Select Technology - Base Frequency (Intel® SST-BF) and Intel® Speed Select Technology – Core Power (Intel® SST-CP) support
- Intel® Software Guard Extensions (Intel® SGX) support
- Intel® Ethernet Controller E810 Series Adapter Support
- FW Update Support for Intel® E710 and E810 Series Adapters
- Location-based Configuration Profiles
- Kubernetes version update
- Kubernetes feature/plugin updates (CMK, Multus, QAT, TAS, SR-IOV, and the like)
- Intel® Dynamic Device Personalization (DDP) support for E710 and E810 Series Adapters
- CentOS 7.6 RT kernel installation
- Introduction of Telemetry components (collectd, Prometheus, node-exporter, Grafana)
- Kubernetes Pod Security Policies and a more secure container registry

G.10 BMRA 2.0 Bug Fixes

The following bug fixes were completed in the BMRA 2.0 release:

- Fixed deployment issues with Ubuntu 18.04 nodes
- Updated CMK to v1.5.1 to address pod restarting issues
- Fixed SR-IOV VF bindings with 710 Network Adapters
- General network driver installation fixes
- Fixed connection issues between pods on different nodes

G.11 Known Issues

Issue:

Occasionally the sriov-network-device-plugin does not detect new or updated VF resources.

Detail:

There is a known issue with sriov-network-device-plugin where the service fails to detect new or updated VF resources if not available when the service creates its ConfigMap and loads the daemonset. See https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin/issues/276.

Workaround:

Delete the sriov-device-plugin-pod and resources will be present when pod is automatically restarted.

Issue:

Intel® Speed Select tool errors on 3rd Generation Intel® Xeon® Scalable processor servers with RHEL 8.2.

Detail:

The currently distributed RHEL 8.2 kernels are not compiled with CONFIG_INTEL_SPEED_SELECT_INTERFACE enabled.

Workaround:

Upgrade to RHEL 8.3.

The RHEL 8.2 default kernel can be recompiled with this setting (check your vendor support before proceeding). Alternatively, the Intel SST-BF feature is available on selected 2nd Generation Intel® Xeon® Scalable processor SKUs (https://access.redhat.com/articles/4481221) and both the Intel SST-BF and Intel SST-CP features are available on select 3rd Generation Intel® Xeon® Scalable processor SKUs with other supported OSes.

Issue:

KMRA PCCS pod fails to load on Ubuntu 20.04.

Detail:

The PCCS container requires kernel 5.4.0-65 or greater.

Workaround:

Update your Ubuntu 20.04 release to Ubuntu 20.04.2 or newer.

Issue:

Collectd pod fails to start on Ubuntu 18.04 inbox kernel.

Detail:

```
intel_rapl: driver does not support CPU family 6 model 106
```

The Intel RAPL power capping driver in the 18.04 inbox kernel does not support 3rd Generation Intel® Xeon® Scalable processors.

Workaround:

Use update_kernel in Ansible group_vars or install a more recent OS with a newer kernel.

Issue:

Collectd plugin fails to start.

Detail:

On some platforms the collectd pod fails to start due to various plugin incompatibilities.

Workaround:
Disable problematic collectd plugins by adding to the exclude_collectd_plugins list in the Ansible host_vars configuration file.

Part 5:
Appendix H
Workloads and Application Examples

Appendix H Workloads and Application Examples

This appendix provides examples of how to provision and deploy example applications or workloads.

H.1 Enabling Key Management NGINX Applications

To set up KMRA infrastructure and run the Key Management workload, follow the steps below. KMRA currently supports Ubuntu 20.04, SGX version 2.12, and DCAP 1.9.

The BMRA infrastructure sets up KMRA distributed HSM key server and compute nodes with SGX. A private key is securely provisioned to Intel Crypto-api-toolkit on the compute node and imported into a token named kmra_token. The provisioned key is named nginx_hsm_priv and the pin is 1234. When configuring a workload to use a private key from Intel Crypto-api-toolkit, the workload (in this demo the workload is NGINX) is configured with a URI for the private key.

The URI used in this demo is `engine:pkcs11:pkcs11:token=kmra_token;object=nginx_hsm_priv;pin-value=1234`

The Key Management NGINX application below sets up NGINX workload and configures it with OpenSSL. A custom version of OpenSSL is installed on the compute node and configured with the libp11 interface and pkcs11 engine. The pkcs11 engine is an interface to Intel Crypto-api-Toolkit with SGX, and it uses the URI configured in the NGINX workload to use the private key. A certificate is generated and signed inside the Intel Crypto-api-Toolkit enclave and stored on the compute node. A link to the certificate is added to the NGINX configuration. The last step of the workload setup is a test using opens s_time to verify the number of TLS connections that NGINX is able to establish using the private keys from Intel SGX enclave.

1. Download KMRA source code.
    a. Go to https://01.org/key-management-reference-application-kmra and download Key Management Reference Application (KMRA) v1.2.1 source code package.
    b. Create KMRA folder in root directory of BMRA.
    c. Untar and place contents into KMRA directory.
2. Run BMRA Configuration Profile full_nfv with some variable changes in group_vars/all.
    a. Kubernetes option must be set to false.
    b. Proxy settings must be changed accordingly.
    c. Set `kmra_use_custom_package_versions` to `true`.
3. Go to KMRA folder created in step 1 and run the NGINX workload Ansible script.
    a. Go to KMRA/ansible/sgx_infra_setup.
    b. Update inventory file with correct key server and compute node hosts.
    c. Run Ansible scripts using the following command:
    ```
    ansible-playbook -i inventory provision_ctk_token_and_start_nginx.yml
    ```

1021/DN/WIT/PDF 632290-007US