Solution Brief

Kubernetes - Cloud Native Orchestration

How Intel makes underlying capabilities in the infrastructure available for consumption by applications.

Executive Summary

This document describes the various types of capabilities, features, and tools Intel provides to enhance the performance of workloads and how Intel makes those capabilities available for easy consumption by cloud native applications.

This document is part of the Network & Edge Platform Experience Kits.

Introduction

In the last few years, cloud adoption triggered application evolution from monolithic apps to highly disaggregated microservices. This document explains the motivation for such transformation and some of the challenges to orchestrating microservices.

Next, we discuss the role of Kubernetes in cloud native orchestration and how it can be used to manage microservices. An overview of the various types of acceleration devices/features Intel provides with its compute and network platforms follows. Finally, we discuss how Intel enables underlying capabilities in Kubernetes to simplify and optimize workload performance in a cloud native environment.

Transition to Containers and Microservices

A key element of the cloud native environment is the disaggregation of workloads from monolithic to their atomic components (called microservices).

Traditional monolithic applications contain all the logic required to deliver a service, running as a single process. This includes generic functionality such as database access or security. While functional elements in the application vary in scale, a monolithic workload's scale is limited by the lowest performing element, resulting in stranded resources, inefficient usage of the infrastructure, and increased costs. The scale of a service deployment is therefore typically determined statically. Scale can only change by adding or removing instances of the hardware and application combination, which is coarse grained and service disruptive.

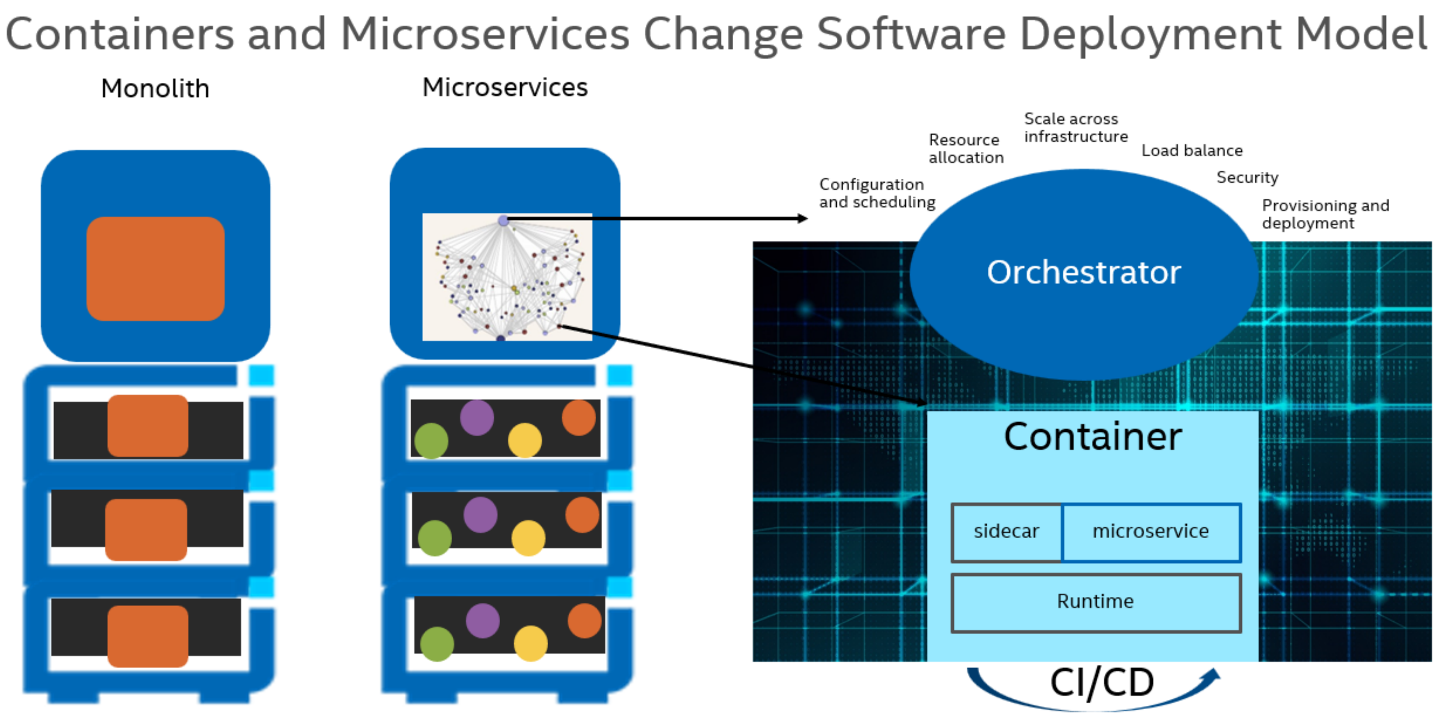
Containers and Microservices Change Software Deployment Model
Monolith
Microservices
Scale across infrastructure
Resource allocation
Load balance
Security
Configuration and scheduling
Provisioning and deployment
Orchestrator
Container
sidecar
microservice
Runtime
CI/CD

Figure 1. Containers and Microservices Change Software Deployment Model

In a cloud native environment, services are decomposed into the small, atomic functions (e.g., database lookup, table search, encryption) running independently and communicating with each other. A service is a loose framework of (containerized) microservices operating in concert. And scalability is achieved by dynamically adding or removing specific microservices as necessary.

The Benefits of Microservice Architectures

The key benefits of microservices architectures are:

1. **Business Agility.** Microservices support the DevOps mindset, allowing business to rapidly develop and deploy new features and services. Reuse and sharing of common functionality between services saves development time and effort. Microservices functionality can be upgraded independently and asynchronously.
2. **Scalability.** Easy to add, remove, update, and scale individual microservices in live production networks. Care is taken to use stateless microservices as much as possible, resulting in seamless upgrades that minimize service interruption.
3. **Efficiency.** Only microservices affected by demand fluctuations need to scale, rather than the entire service, resulting in optimal resource utilization and reduced costs.
4. **Resilience.** Failures are confined to individual microservices and only affect the overall service marginally. Failed microservice instances can quickly and automatically be replaced with new instances.
5. **Reusability.** Microservices can be used for many purposes in multiple services.

Given the benefits of microservices, it is clear why, as research shows, microservices adoption is high among the leaders in cloud native transformation and their use is growing in various industries.

The Challenges in Using Microservices

Microservices substitute large, monolithic, and cohesive, hardwired applications with large numbers of small, distributed, and dynamic functions. To make microservices work, automated orchestration is imperative. Human interaction cannot sustain the speed and scale of the cloud.

A microservices orchestration solution must automatically identify resources and capabilities available in the infrastructure, dynamically reserve and configure such resources as the applications require, and place the applications where the required resources have been reserved.

Another challenge is the increased complexity and importance of monitoring, given the scale and dynamism of cloud native deployments.

Kubernetes: High-Level Architecture and Functionality

The increasing adoption of cloud native practices gave rise to Kubernetes as the primary application orchestration platform for the cloud. Kubernetes (K8s), which is sometimes referred to as the 'Operating System for the Cloud', addresses the challenges. It provides key services, such as scheduling or instantiating VMs and containers on demand and placing workloads in a cluster of nodes.

As the leading open-source cloud orchestration platform, Intel invests in technology development and contributes to Kubernetes. Primary areas of focus are:

- Resource management or locating and optimally allocating metered resources to workloads.
- Scaling and self-healing of cloud native applications.
- Load balancing or efficiently distributing load across the cluster.
- Enablement of infrastructure capabilities.

Solution Description

Objectives of Enablement of Intel Capabilities

Intel develops advanced tools and capabilities to enhance the performance of its network infrastructure products (compute, network, and storage) when deploying applications and services in a cloud environment.

Enabling such capabilities encompasses exposing them to applications through Kubernetes and equipping Kubernetes with the tools to reserve and allocate such capabilities efficiently as required by the workloads.

To expose capabilities to the applications, Kubernetes needs to discover and map the available capabilities. To that end, a function named Node Feature Discovery runs on each node, detects which capabilities (out of a predetermined list) exist on the node, and advertises the capabilities to the Kubernetes control plane. Kubernetes then can reserve and allocate the resources before placing (scheduling) workloads that require them.

In addition to matching workload resource requests with available resources and capabilities, Kubernetes could use granular understanding of the internal architectures of the hardware to further optimize resource reservation and allocation.

Intel provides this functionality for its capabilities, to simplify and enhance deployments over its platforms. This is done in open source and in a hardware agnostic manner, such that Intel's competitors are free to enable their respective capabilities.

Areas of Focus for Capability Enablement in Kubernetes

Intel's Kubernetes enhancements could be classified into the following areas of focus:

- **Kubernetes Networking Enhancements** that enable Intel® Ethernet features and enable advanced networking features.
- **Packet Processing Acceleration** enabling features that enhance data plane performance.
- **Management and Scheduling**, which enable optimized resource allocation.
- **Accelerator Device enablement**, exposing and allocating Intel accelerator devices and features.

Types of Capabilities Enabled

Intel offers a variety of capability types on its platforms, each with their own enablement methods in Kubernetes.

1. **Specialized x86 Instruction Sets.** These are extensions to the basic x86 instruction set to accelerate specific operations. For example, Intel® Advanced Vector Extensions (Intel® AVX) accelerates floating point operations. Applications in video processing, simulations, 3D modeling, and the like, can be floating point intensive and benefit from Intel AVX.
2. **Software Technologies**, such as Intel® Software Guard Extensions (Intel® SGX), to enhance application security.
3. **Embedded Hardware Accelerators** offload certain operations such as streaming and free up processor cycles for other tasks.
4. **Intel Field Programmable Gate Array (FPGA) Devices.** These are programmable devices that could be flexibly applied to perform functions, such as Forward Error Correction, to offload the processor.
5. **PCIe Devices** such as network interface cards, data encryption accelerator cards, like Intel® QuickAssist Technology (Intel® QAT) and graphics processing units (GPUs).
6. **Intel Storage and Memory Devices.**

Technologies Implemented

Enabling Instruction Sets in Kubernetes

Instruction sets are groups of commands for a CPU in machine language. Special instruction sets accelerate CPU performance for certain use cases (e.g., multimedia, scientific, financial applications).

New computer platforms often introduce new special instruction sets. Because instruction sets are not metered resources, enablement is simply via Node Feature Discovery. Node Feature Discovery scans the server for special instruction sets and advertises those that are available.

Enabling Acceleration Devices

New Intel compute platforms come with embedded hardware accelerators, designed to offload certain functions from the CPU. These might include, for example, compression and encryption. Acceleration devices offer performance gains for the CPU as they free up CPU cores for other uses. Unlike special instruction sets, these hardware accelerators are typically metered. This means that device capacity can be apportioned to multiple workloads.

Metered resources require device drivers for the operating system to access. In some cases, Intel provides out-of-tree (OOT) device drivers for early access to new functionality, while in-tree drivers are making their way into the Linux kernel and into commercial operating system distributions.

Hardware accelerators and other metered resources require Kubernetes device plugins for setup, configuration, and life cycle management. Intel provides device plugins for its acceleration devices, as well as automated tools to manage the life cycle of OOT device drivers and of Kubernetes device plugins.

Intel also builds frameworks for telemetry collection and observability of resources.

Device Plugins

Device plugins are custom Kubernetes code that device vendors provide to enable integration and allocation of metered resources. However, because they are provided by multiple vendors and use custom code, it is difficult for customers to consume these plugins.

To make this task easier, Intel provides a device plugin framework that enables vendors to advertise, schedule, and set up devices natively in Kubernetes. This makes device plugins easy to deploy via Kubernetes and workloads can request devices via extended resource requests in the pod specification.

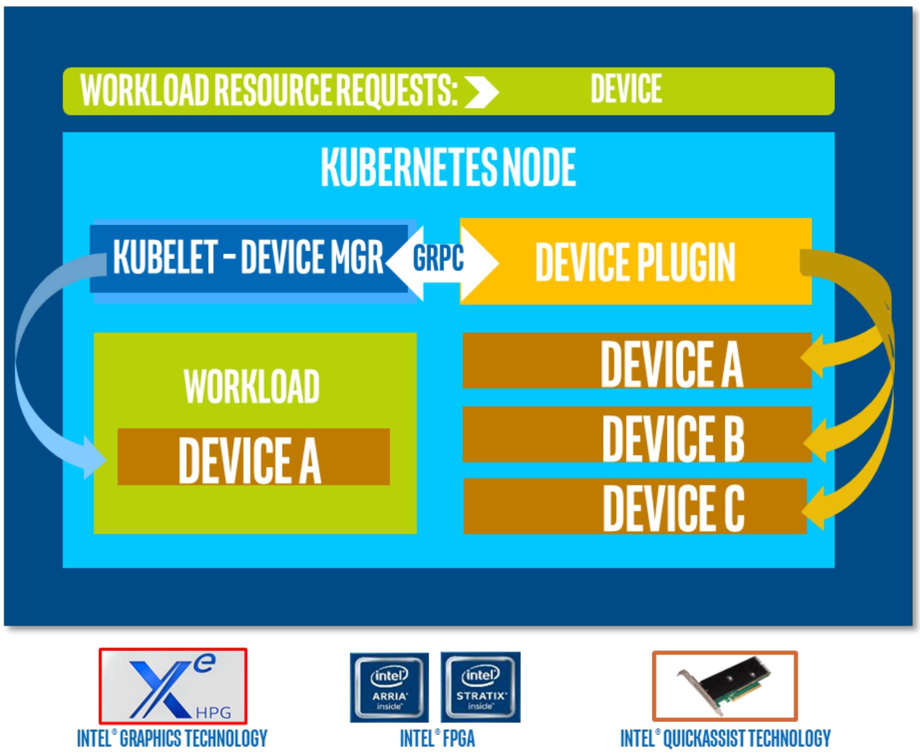

Figure 2. Device Plugin Framework

References:

- SR-IOV Network Device Plugin, (V3.3.1, January 2021)
- Collection of Intel Device Plugins (Intel QAT and FPGA), (V0.2, March 2021)
- Device Plugins Documentation

Node Feature Discovery

There is no way in core Kubernetes to identify hardware capabilities on nodes in the network. Node Feature Discovery detects capabilities on each node in a Kubernetes cluster and advertises them to the control plane. This allows Kubernetes to match a workload to the platform capabilities.

References:

- Node Feature Discovery, (V0.10.1, January 2022)
- Node Feature Discovery Application Note

Telemetry Collection and Observability

Platform health information is key to several systems, including high availability and resiliency systems, root cause analysis systems, and SDN systems. Telemetry is also important for correlating network conditions with platform processing conditions – enabling optimized scheduling. Also, telemetry allows the creation of predictive models and the development of preventative maintenance solutions.

Platform utilization/capacity reports can be used to detect platforms trending to resource exhaustion, before the workload experiences issues that could affect service. This allows remediation and rebalancing of traffic. Insights can be used to indicate overload on the CPU, congestion on the interfaces, and can help identify configuration errors before they cause issues for the service or application.

Lifecycle Automation and Kubernetes Operators

Several widely accepted technologies exist to automate the deployment and configuration of complex resources, for example,

- Helm: Set of tools to deploy resources
- Ansible: Creates scripts for provisioning and configuring resources

If the desired resource state gets misaligned with the actual resource state, a 'controller' is needed to bring them back into alignment. A common format for such controllers is the Kubernetes operator. Intel provides several resource operators to simplify the deployment and configuration of certain capabilities.

Use Case Examples

Networking Example - Multus

Kubernetes supports only a single network interface per pod and NFV use cases often require multiple network interfaces to be available to the virtualized operating environment of the VNF. The solution shown here uses Multus, a CNI proxy and arbiter of other CNI plugins. Multus invokes other CNI plugins for network interface creation like the main plugin, which is identified to manage the primary network interface (eth0) for the pod. Other CNI minion plugins, such as SR-IOV, vHost CNI, and more, can create additional pod interfaces during their normal instantiation process.

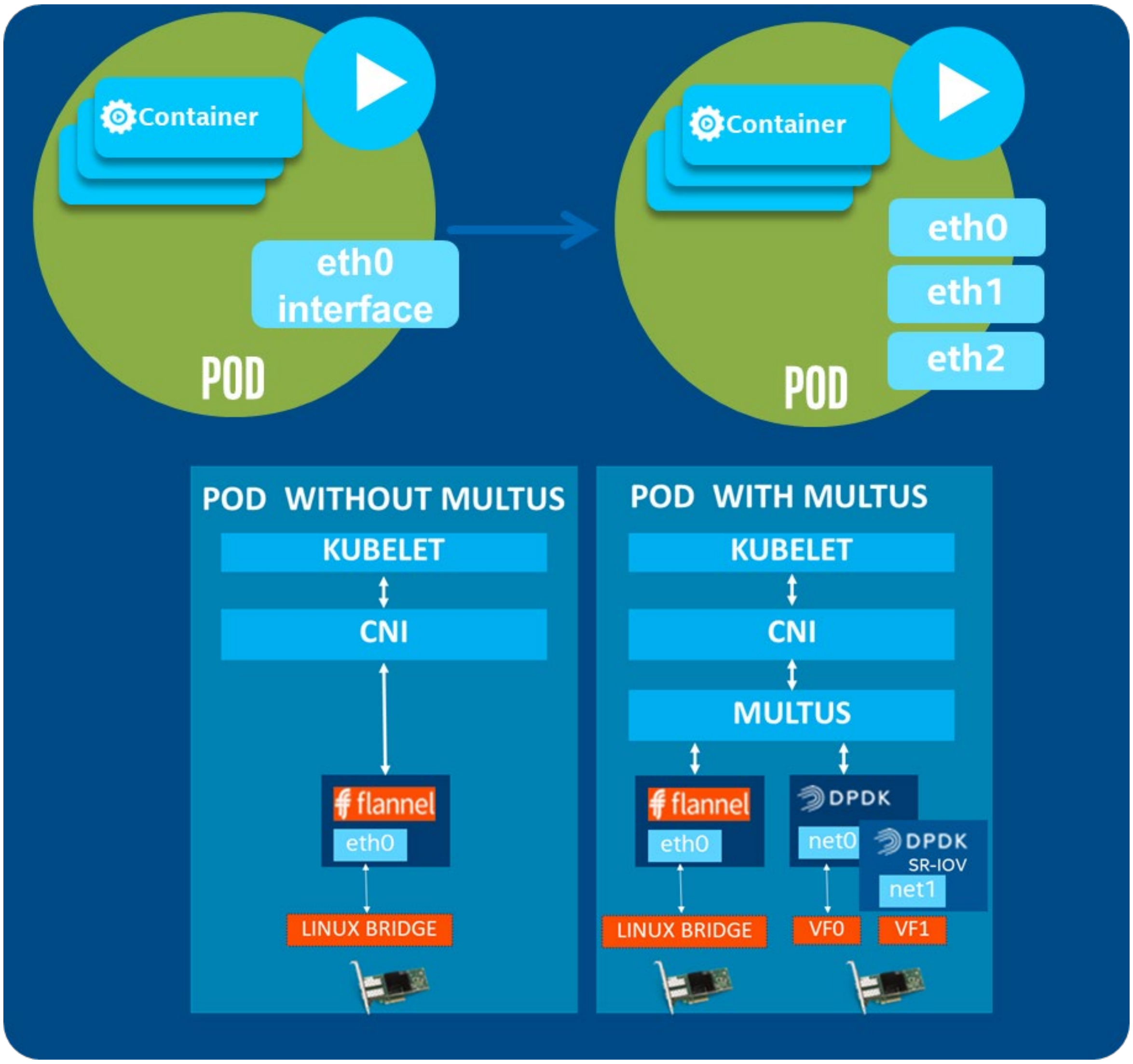

Figure 3. Multus Example

References:

- K8s Network Plumbing Working Group Spec
- Multus CNI, (v3.7.1 March 2021)

The benefits of Multus include network segregation for functional purposes such as to improve performance, network segregation for non-functional purposes such as to improve security, and link aggregation or bonding for network interface redundancy.

Life Cycle Automation Example – Intel® Ethernet Operator

Cloud admins want to be able to configure and deploy firmware and driver updates at scale – and in an automated manner. There is also a need in private cloud for workload specific network optimization using the Intel® Ethernet 800 Series Network Adapter's advanced Dynamic Device Personalization (DDP) capability in a Kubernetes cluster. The Ethernet Operator provides an overall framework to enable firmware and DDP using an operator construct. This delivers workload networking optimization at the admin level for Intel® Ethernet Network Adapters at cluster level.

Reference:

- Intel Ethernet Operator

Resource Management Example

The Kubernetes CPU Manager and Device Manager make resource allocation decisions independently of each other. This can negatively impact application performance/latency.

Solution:

- Topology Manager, a kubelet component, is an interface for coordinating resource assignments on a node level.
- Manages the resources allocated to workloads in a NUMA topology-aware manner.

Kubernetes CPU Manager and Device Manager are the first components to implement the Topology Manager interface.

Benefit/Usage: Allows coordinated resource allocation on the node level.

References:

- Kubernetes Topology Manager, (Beta status since K8s V1.18)
- Kubernetes Blog
- Topology Management – Implementation in Kubernetes* Technology Guide
- Topology Management in Kubernetes – Training Video

The following diagram illustrates resource allocation decision optimization with the Topology Manager, as compared to the decision made without the Topology Manager:

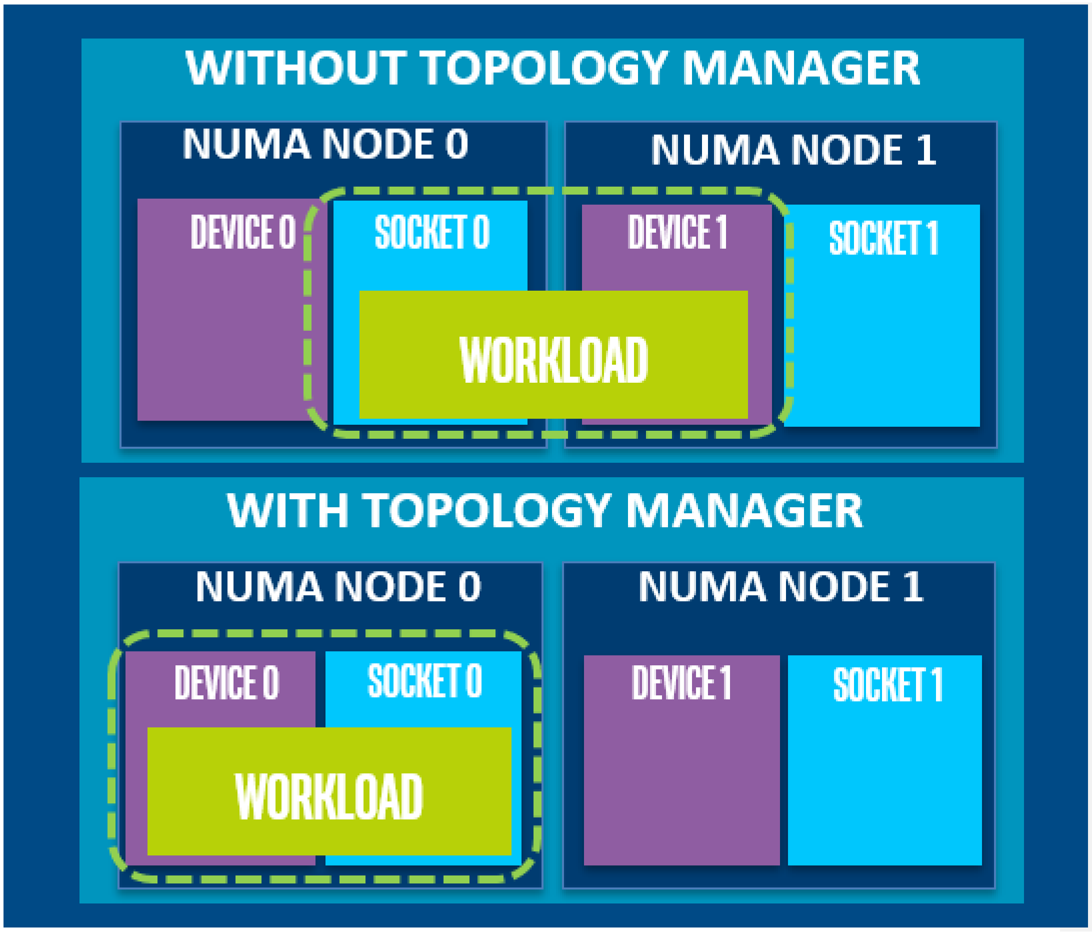

Figure 4. Resource Management With and Without Topology Manager

Accelerator Example

It could be difficult to match applications that require cryptography or compression with platforms with the Intel® QuickAssist Technology (Intel® QAT). In the example shown here, Intel QAT support is enabled through a dedicated device plugin.

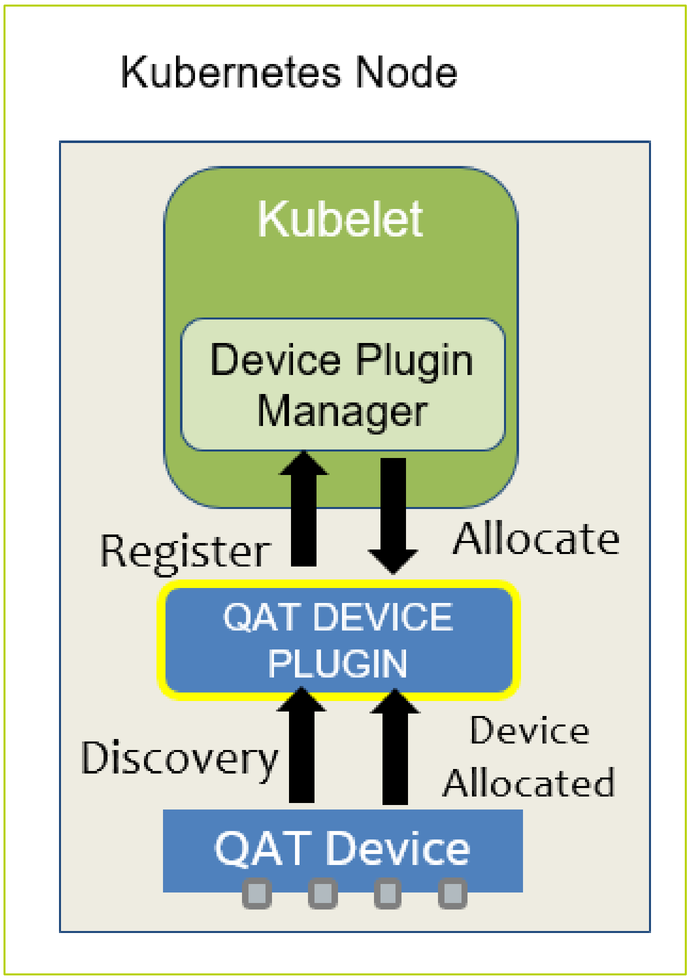

Figure 5. QAT Accelerator Enablement Example Using Device Plugin

References:

- Intel QAT Device Plugins for Kubernetes
- Intel Device Plugins for Kubernetes* Application Note

The Intel QAT device plugin discovers Intel QAT support on a node and the number of virtual functions that are configured. It then advertises this to the node and allocates virtual functions based on workload resource requests.

Summary

In this document, we explain the evolution of cloud native applications from monolithic form to disaggregated, distributed microservices form and the challenges that this transformation presents. We explained the various enhancements Intel provides with its products, and how such contributions are exposed and enabled through Kubernetes for ease of adoption by cloud native workloads.

Intel capabilities are designed to improve the performance of cloud native workloads when running on Intel platforms. Due to space restrictions, this document only provides a few examples of how Intel enables its technology for the cloud. It is recommended to those interested to engage with Intel for further and deeper understanding, as well as for feedback and specific use case discussions.

Terminology

Table 1. Terminology

Abbreviation	Description
CNI	Container Network Interface
DDP	Dynamic Device Personalization
FPGA	Field Programmable Gate Array
gRPC	Remote Procedure Call
K8s	Kubernetes
NFV	Network Functions Virtualization
NUMA	Non-Uniform Memory Access
OOT	Out of Tree
SDN	Software-Defined Networking
SR-IOV	Single Root I/O Virtualization
VM	Virtual Machine
VNF	Virtual Network Function

Document Revision History

Revision	Date	Description
001	January 2023	Initial release.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

0123/DN/WIT/PDF 764908-001US