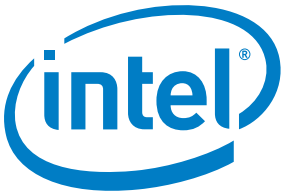


Intel Device Plugins for Kubernetes*

Application Note

December 2018

Document Number: 606832-001

Contents

Figures

Revision History

Date	Revision	Description
December 2018	001	Initial release.

1.0 Introduction

Many vendors, including Intel, have created hardware devices that deliver efficient acceleration of graphics, computation, data processing, security, and compression. These devices provide optimized hardware for specific tasks, which saves CPU cycles for other workloads and typically results in performance gains. However, device vendors must then write custom code to integrate their device in the container ecosystem.

The Kubernetes* device plugin framework provides a vendor-independent solution for hardware devices. Intel has developed a set of device plugins that comply with the Kubernetes* device plugin framework and allow users to request and consume hardware devices across Kubernetes clusters.

This document is targeted for a technical audience, including developers looking to efficiently support hardware devices in container environments. It provides an overview of the following newly-introduced open source device plugins:

- Graphics Processing Unit (GPU) plugin
- Field Programmable Gate Array (FPGA) plugin
- Intel® QuickAssist Technology (QAT) plugin

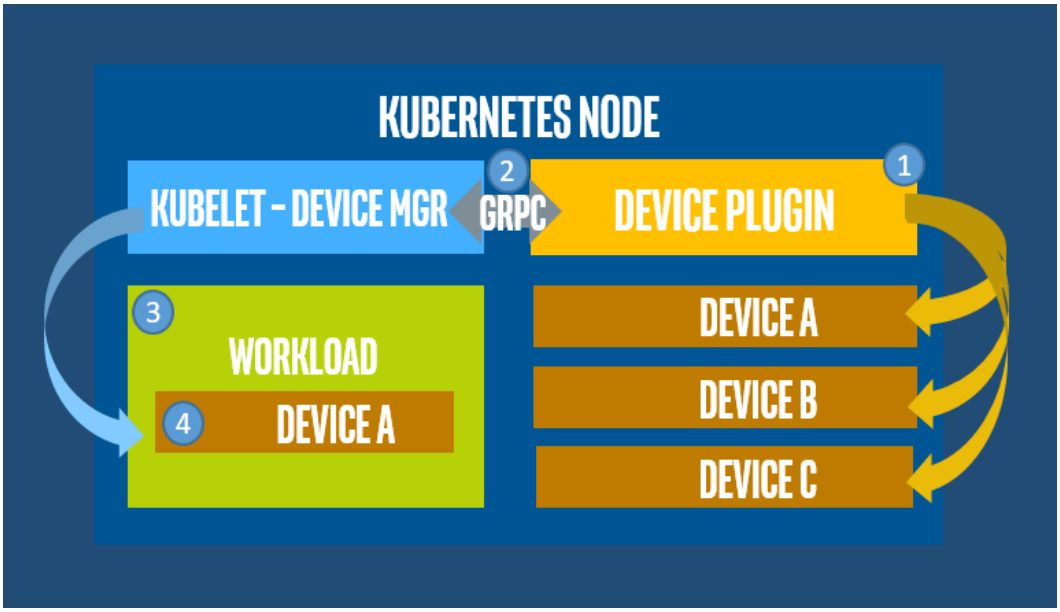
Detailed configuration and deployment instructions for the plugins is available at: https://github.com/intel/intel-device-plugins-for-kubernetes/blob/master/README.md

2.0 Device Plugin Overview

The following figure shows how device plugins operate on a Kubernetes node:

1. During setup, the cluster admin knows what kind of devices are present on the different machines. The cluster admin selects which devices to enable and deploys the associated device plugin.
2. The plugin reports the devices it found on the node to the Kubelet device manager and starts its gRPC server to monitor the devices.
3. A user submits a pod spec (workload) requesting a certain type of device.
4. Kubelet decides which device to assign to the pod's containers.

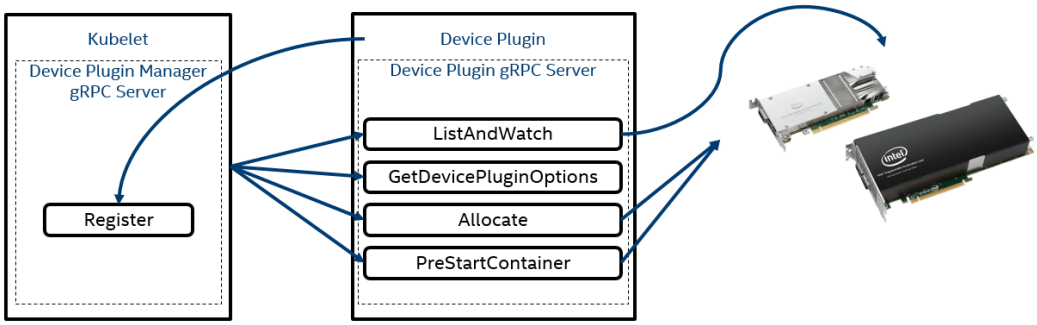
Figure 1. Kubernetes* Device Plugin Operations



Device plugin implementation is shown in the following figure and organized as follows:

- Register: The device plugin advertises its presence to Kubelet using gRPC on an Unix socket.
- ListAndWatch: The device plugin advertises a list of devices to Kubelet. It resends the list if the state of a device changes.
- GetDevicePluginOptions:
  GetDevicePluginOptions returns options to be communicated with Device Manager. Currently only one option is supported (`pre_start_required`), which indicates if a PreStartContainer call is required before each container start.
- Allocate: When creating containers, Kubelet calls the device plugin's allocate function so that it can run device-specific instructions (such as gpu cleanup, QRNG initialization, etc.) and tell Kubelet the steps to make the device available.
- PreStartContainer: When a device request is received, Kubelet makes the device available in the container.

Figure 2. Kubernetes* Device Plugin Implementation



For more information about Kubernetes device plugins, refer to:

- https://kubernetes.io/docs/concepts/cluster-administration/device-plugins/
- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/resource-management/device-plugin.md

3.0 Graphics Processing Unit (GPU) Plugin

The Intel GPU plugin supports the following devices:

- Integrated GPUs within Intel® Core™ and Intel® Xeon® processors
- Intel® Visual Compute Accelerator (Intel® VCA)

The GPU plugin offloads the processing of computation intensive workloads to GPU hardware. There are two primary use cases: one is hardware vendor-independent and uses the Intel Media SDK, and the other uses OpenCL™ code tuned for high end Intel devices. For example, the Intel Media SDK can offload video transcoding operations or the OpenCL™ libraries can provide computation acceleration for Intel GPUs.

The GPU plugin demo shows the deployment of the Intel® GPU Device Plugin for Kubernetes including Kubeless* Function as a Service (FaaS) media transcoder JavaScript* function. The media transcoding workload is scheduled on two different worker nodes, but only one worker node has a GPU device. The demo captures the time difference in transcoding speed for comparison. The demo includes the following steps:

1. Validate the status of the Kubernetes cluster.
2. Install the Kubeless environment, Apache Kafka*, Apache Zookeeper*, and Minio*.
3. Connect Minio and Apache Kafka.
4. Build the Kubeless runtime with Intel® Media Driver for VAAPI accelerated ffmpeg.

5. Add the new runtime to the Kubeless config map.
6. Run the GPU device plugin.
7. Deploy the function, capture data, and review the logs to evaluate performance.

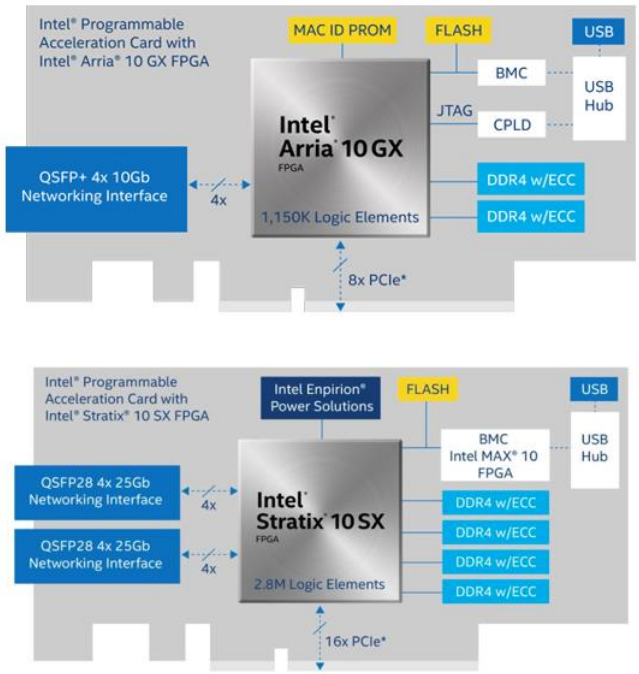
For more information, refer to:

- Plugin build and test details: https://github.com/intel/intel-device-plugins-for-kubernetes/blob/master/cmd/gpu_plugin/README.md
- Demo details: https://github.com/intel/intel-device-plugins-for-kubernetes/tree/master/demo
- Intel® Visual Compute Accelerator: https://www.intel.com/content/www/us/en/products/servers/accelerators.html

4.0 Field Programmable Gate Array (FPGA) Plugin

The Intel FPGA plugin supports the Intel® Programmable Acceleration cards with Intel® Arria® 10 FPGA and Intel® Stratix® 10 FPGAs, which are shown below:

Figure 3. Devices Supported by FPGA Device Plugin



The FPGA plugin supports the Open Programmable Acceleration Engine (OPAE) framework and OpenCL™ code. It is designed for pre-programmed accelerators, orchestration programming, and access control.

The FPGA plugin is comprised of the following modules:

- **FPGA device plugin** is responsible for discovering and reporting FPGA devices to the Kubelet. During the allocation phase, it instructs Kubelet about device nodes to be accessed by the container and sets the annotation that is later on passed to CRI to trigger programming by the CRI-O hook.
- **FPGA prestart CRI-O hook** is triggered by pod annotation set by the FPGA device plugin. It performs discovery of the requested FPGA function bitstream and then programs FPGA devices based on the environment variables in the workload description.
- **FPGA admission controller webhook** is responsible for performing mapping from user-friendly function IDs to the Interface ID and Bitstream ID that are required for FPGA programming CRI-O hook. Mappings are stored in namespaced custom resource definition (CRD) objects, therefore the admission controller also performs access control, determining which bitstream can be used for which namespace. The admission controller also keeps the user from bypassing namespaced mapping restrictions, by denying admission of any pods that are trying to use internal knowledge of Interface ID or Bitstream ID environment variables used by the prestart hook.

The Intel® FPGA Device Plugin demo shows the deployment of the Intel® FPGA Device Plugin for Kubernetes and executes a sample GZIP compression workload. The demo begins with a fully configured Kubernetes cluster with the Go runtime and continues with the following steps:

1. Validate the status of the Kubernetes cluster.
2. Clone the Intel device plugins for Kubernetes source.
3. Provision the admission controller webhook.
4. Provision the Intel® FPGA device plugin.
5. Create bitstream storage for the Intel® FPGA.
6. Run the sample GZIP compression workload.

For more information, refer to:

- Plugin build and test details: https://github.com/intel/intel-device-plugins-for-kubernetes/blob/master/cmd/fpga_plugin/README.md
- Demo details: https://github.com/intel/intel-device-plugins-for-kubernetes/tree/master/demo
- Intel® Programmable Acceleration cards: https://www.intel.com/content/www/us/en/programmable/solutions/acceleration-hub/platforms.html

5.0 Intel® QuickAssist Technology (QAT) Plugin

Intel® QuickAssist adapters integrate hardware acceleration of compute intensive workloads such as bulk cryptography, public key exchange, and compression on Intel® Architecture Platforms.

The Intel® QAT device plugin for Kubernetes supports Intel® QuickAssist adapters and includes an example scenario that uses the Data Plane Development Kit (DPDK) drivers. An additional demo that executes an Intel® QAT accelerated OpenSSL* workload with the Kata Containers runtime is also available.

The DPDK demo performs the following steps:

1. Validate the status of the Kubernetes cluster.
2. Deploy Intel® QAT device plugin directly on the host.
3. Build the DPDK image.
4. Deploy a pod to run an example DPDK application.
5. Manually execute the dpdk-test-crypto-perf application to review the logs and evaluate the data.

The OpenSSL demo performs the following steps:

1. Prepare the host drivers and virtual function (VF) devices.
2. Check the Kubernetes cluster is in good shape.
3. Deploy the Intel® QAT device plugin for Kubernetes.
4. Deploy Intel® QAT Accelerated OpenSSL workload.
5. Test.

For more information, refer to:

- Plugin build, test, and DPDK demo details: https://github.com/intel/intel-device-plugins-for-kubernetes/blob/master/cmd/qat_plugin/README.md
- OpenSSL demo details: https://github.com/intel/intel-device-plugins-for-kubernetes/tree/master/demo
- Intel® QuickAssist adapters: https://www.intel.com/content/www/us/en/ethernet-products/gigabit-server-adapters/quickassist-adapter-for-servers.html