TECHNOLOGY GUIDE

Intel® Corporation

Node Feature Discovery - Orchestrating Intelligent Workload Scheduling

Author

Gershon Schatzberg

1 Introduction

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. This document details the deployment and usage of NFD. It is written for developers and architects who want to integrate NFD into their Kubernetes deployment to facilitate improved workload placement based on platform capabilities.

NFD is an open source Kubernetes community project. The software is available at GitHub*.

Note: This document does not describe how to set up a Kubernetes cluster. We recommend that you perform those steps as a prerequisite.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	December 2018	Initial release.
002	February 2022	Added new features.
003	February 2022	Revised the document for public release to Intel® Network Builders.

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
CPUID	CPU Identification
DPDK	Data Plane Development Kit
EPA	Enhanced Platform Awareness
etcd	Distributed key value store that serves as a reliable way to store data across a cluster of machines
Intel® AVX	Intel® Advanced Vector Extensions
Intel® RDT	Intel® Resource Director Technology
Intel® VT-d	Intel® Virtualization Technology (Intel® VT) for Directed I/O
IOMMU	Input/Output Memory Management Unit
NFD	Node Feature Discovery
NUMA	Non-Uniform Memory Access
PF	Physical Function
P-state	CPU performance state
RDT	Intel® Resource Director Technology
SLA	Service Level Agreement
SR-IOV	Single Root I/O Virtualization
VF	Virtual Function

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
Enhanced Platform Awareness in Kubernetes Application Note	https://networkbuilders.intel.com/solutionslibrary/enhanced-platform-awareness-in-kubernetes-application-note
Enhanced Platform Awareness in Kubernetes Feature Brief	https://networkbuilders.intel.com/solutionslibrary/enhanced-platform-awareness-feature-brief
Enhanced Platform Awareness in Kubernetes Performance Benchmark Report	https://networkbuilders.intel.com/solutionslibrary/enhanced-platform-awareness-in-kubernetes-performance-benchmark-report
Enabling New Features with Kubernetes for NFV White Paper	https://networkbuilders.intel.com/solutionslibrary/enabling-new-features-in-kubernetes-for-nfv

2 Overview

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 detects these features and advertises them through a Kubernetes concept called node labels, which in turn, can be used to control workload placement in a Kubernetes cluster. NFD runs as a separate container on each individual node of the cluster, discovers capabilities of the node, and finally, publishes these as node labels using the Kubernetes API.

NFD only handles non-allocatable features, that is, unlimited capabilities that do not require any accounting and are available to all workloads. Allocatable resources that require accounting, initialization, and other special handling (such as Intel® QuickAssist Technology, GPUs, and FPGAs) are presented as Kubernetes Extended Resources and handled by device plugins. They are out of the scope of NFD.

NFD currently detects the following features:

- **CPUID:** Intel® processors have a special CPUID instruction for determining the CPU features, including the model and support for instruction set extensions, such as Intel® Advanced Vector Extensions (Intel® AVX), CLDEMOTE, ENQCMD, 5G-

ISA, MOVDIR, etc. Certain workloads, such as machine learning, may gain a significant performance improvement from these extensions (for example Intel® AVX-512). NFD advertises all CPU features obtained from the CPUID information.

- **SR-IOV networking:** Single Root I/O Virtualization (SR-IOV) is a technology for isolating PCI Express* resources. It allows multiple virtual environments to share a single PCI Express hardware device (physical function, PF) by offering multiple virtual functions (VF) that appear as separate PCI Express interfaces. In the case of network interface cards (NICs), SR-IOV VFs allow direct hardware access from multiple Kubernetes pods, increasing network I/O performance, and making it possible to run fast user-space packet processing workloads (for example, based in Data Plane Development Kit). NFD detects the presence of SR-IOV-enabled NICs, allowing optimized scheduling of network-intensive workloads.
- **Intel® RDT:** Intel® Resource Director Technology (Intel® RDT) allows visibility and control over the usage 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, RDT helps in mitigating the effects of noisy neighbors. This provides more consistent and predictable performance which, may be essential in meeting Service Level Agreements (SLA), for example. NFD detects the different RDT technologies supported by the underlying hardware platform.
- **Intel® Speed Select Technology:** Intel® Speed Select Technology is a family of features providing nuanced control over CPU. NFD detects the state of Intel® Speed Select Technology, allowing optimal scheduling of workloads that can benefit from 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. This can be crucial for I/O heavy workloads in Kubernetes deployments using hypervisor-based container runtimes, such as Kata* Containers. NFD detects if an IOMMU is supported by the host hardware platform and enabled in the kernel of the host operating system.
- **SSD storage:** Solid state drives (SSD) have a huge performance advantage over traditional rotational hard disks. This may be important for disk I/O intensive workloads. NFD detects the presence of non-rotational block storage on the node, making it possible to accelerate workloads requiring fast local disk access.
- **PMEM storage:** NFD detects the presence of Persistent Memory storage on the node, providing good cost/performance trade-off option.
- **NUMA topology:** Non-uniform memory access (NUMA) is a memory architecture where CPU's memory access times are dependent on the memory location. Access to CPU's local memory is faster than to non-local memory (local memory of another CPU) which can cause workloads to perform poorly if not properly designed for NUMA systems. On the other hand, some highly NUMA-aware applications may experience negligible performance penalties. NFD detects the presence of NUMA topology, making it possible to optimize scheduling of applications based on their NUMA-awareness.
- **Linux* kernel:** Some specific workloads may be highly dependent on the kernel version of the underlying host operating system. For example, some kernel features may be required to be able to run an application, or they provide measurable performance benefits. NFD detects the kernel version and advertises it through multiple labels, allowing the deployment of workloads with different granularity of kernel version dependency.
- **PCI:** Detecting the presence of compatible PCI hardware devices is beneficial for some workloads. For example, Kubernetes device plugins need to be deployed only on nodes that have hardware that the device plugin manages. NFD detects PCI devices, such as Intel® Dynamic Load Balancer (Intel® DLB), allowing optimized scheduling of workloads dependent on certain PCI devices.
    - Intel® Software Guard Extensions (Intel® SGX) helps protect data in use via unique application isolation technology.
    - Intel Dynamic Load Balancer (Intel DLB) is a PCIe accelerator designed for use with DPDK for load-balanced, prioritized scheduling of core-to-core communication, and provides enhanced performance than software-based load balancing.

NFD is under active development, and has evolved significantly, gaining new functionality, including the discovery of multiple new features, as shown in this feature label.

2.1 Technology Description

Node Feature Discovery is designed to run on every node in a Kubernetes cluster, either as a daemon set or a job. The Node Feature Discovery pod discovers capabilities on each node it runs on, and then advertises those capabilities as node labels. As seen in Figure 1 below, NFD has run on each node and discovered the capabilities of the nodes (Turbo Boost, AVX, IOMMU, etc.) and then advertised those as labels, which are stored on the Kubernetes Master node in the ETCD data store. ETCD stores configuration information for large scale distributed systems; the name originates from the `Unix*/etc` folder plus `d` for distributed systems.

Using a Node Selector, an incoming pod can express its requirements for specific capabilities. Figure 1 shows Application A needs to land on a node with SR-IOV and Turbo Boost capabilities. The Kubernetes scheduler on the master node will use the stored node labels to match the incoming pod to the most appropriate node. In the figure, this is Node 1. Application B has no special capability requests; therefore it can be placed on either node. In the figure, it is placed on Node 2.

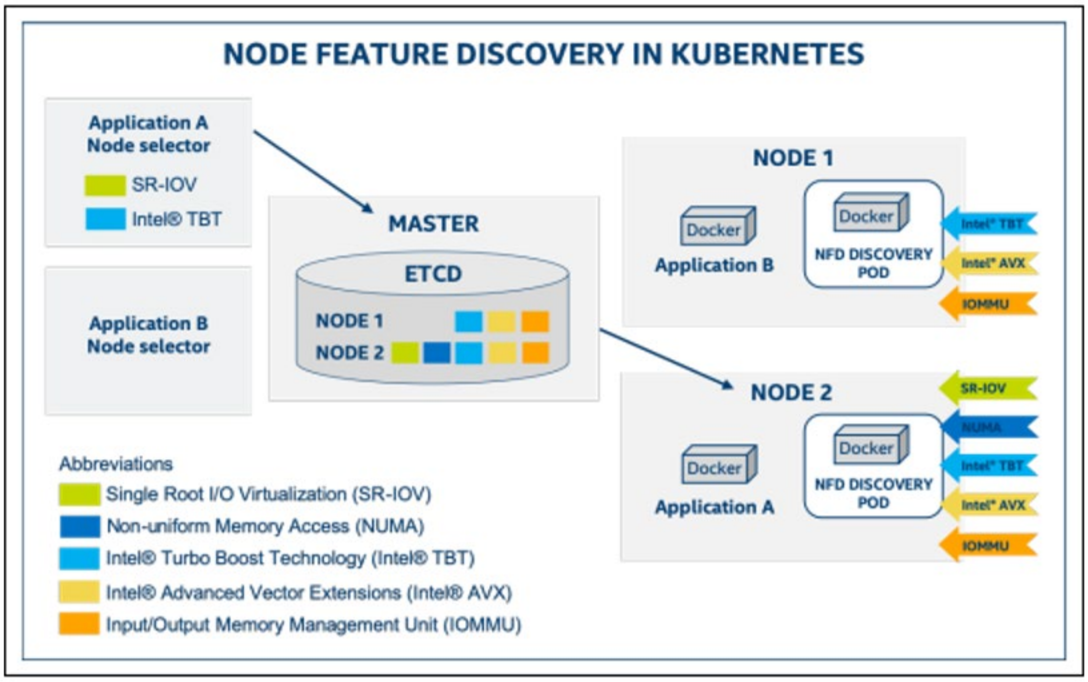
NODE FEATURE DISCOVERY IN KUBERNETES
Application A Node selector
SR-IOV
Intel® TBT
Application B Node selector
MASTER
ETCD
NODE 1
NODE 2
NODE 1
Docker
Application B
Docker
NFD DISCOVERY POD
Intel® TBT
Intel® AVX
IOMMU
NODE 2
Docker
Application A
Docker
NFD DISCOVERY POD
SR-IOV
NUMA
Intel® TBT
Intel® AVX
IOMMU
Abbreviations
Single Root I/O Virtualization (SR-IOV)
Non-uniform Memory Access (NUMA)
Intel® Turbo Boost Technology (Intel® TBT)
Intel® Advanced Vector Extensions (Intel® AVX)
Input/Output Memory Management Unit (IOMMU)

Figure 1 Node Feature Discovery in Kubernetes

2.2 Feature Labels

NFD uses labels for advertising node-level features. Kubernetes labels are key-value pairs that are attached to Kubernetes objects, such as pods or nodes for specifying attributes of objects that may be relevant to the end user. They can also be used to organize objects into specific subsets. Labels are a part of the metadata information that is attached to each node's description. All this information is stored in etcd in the Kubernetes control plane. Node labels published by NFD encode the following information:

- A namespace
- The source of the feature
- The name of the feature, with optional attribute name or sub-feature separated by a dot
- The value or the state of the feature

An example of a label created by node feature discovery:
`node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true`

This indicates that the namespace is `node.alpha.kubernetes-incubator.io`, the source is `network`, feature name is `sriov.capable` and the value is `true`, indicating the presence of SR-IOV capable network interface card.

In addition to the actual node feature labels, NFD advertises its own software version:

```
node.alpha.kubernetes-incubator.io/node-feature-discovery.version =
v0.3.0
```

The following table describes the details of the supported feature sources and their feature labels.

Table 3. Feature Labels

Source	Feature	Attribute	Possible Values	Description
cpuid	<cpuid feature name>	n/a	true	All CPU features returned by the CPUID instruction. Examples: node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX= true node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512F= true node.alpha.kubernetes-incubator.io/nfd-cpuid-SHA= true
kernel	version	full	<version string>	Full kernel version. Example: node.alpha.kubernetes-incubator.io/nfd-kernel-version.full =4.5.6- 7-g123abcde
		major	<version number>	First component of the kernel version. Example: node.alpha.kubernetes-incubator.io/nfd-kernel-version.major= 4
		minor	<version number>	Second component of the kernel version. Example: node.alpha.kubernetes-incubator.io/nfd-kernel-version.minor= 5
		revision	<version number>	Third component of the kernel version. Example: node.alpha.kubernetes-incubator.io/nfd-kernel-version.minor= 6
iommu	enabled	n/a	true	An IOMMU is present and enabled in the kernel. Example: node.alpha.kubernetes-incubator.io/nfd-iommu-enabled =true
memory	numa	n/a	true	NUMAtopology detected. Example: node.alpha.kubernetes-incubator.io/nfd-memory-numa =true
network	sriov	capable	true	SR-IOV capable Network Interface Card(s) present. Example: node.alpha.kubernetes-incubator.io/nfd-network.sriov.capable = true
		configured	true	SR-IOV Virtual Functions have been configured. Example: node.alpha.kubernetes-incubator.io/nfd-network.sriov.configured = true
pci	<device label>	present	true	Presence of PCI device is detected. Example: node.alpha.kubernetes-incubator.io/nfd-pci-1200_8086.present=true

Source	Feature	Attribute	Possible Values	Description
NOTE: **<device label>** is composed of raw PCI IDs, separated by underscores. The set of fields is configurable, valid fields being **class**, **vendor**, **device**, **subsystem_vendor** and **subsystem_device** (defaults are **class** and **vendor**). Also the set of PCI device classes that the feature source detects is configurable. By default, device classes **03(h), 0b40(h), and 12(h)**, i.e. GPUs, co-processors, and accelerator cards are detected. See Runtime Configuration section for more details about NFD configuration options.				
rdt	RDTCMT	n/a	true	Intel® RDT Cache Monitoring Technology is supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTCMT =true
	RDTMBM	n/a	true	Intel® RDT Memory Bandwidth Monitoring is supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBM =true
	RDTMBA	n/a	true	Intel® RDT Memory Bandwidth Allocation is supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBA =true
	RDTMON	n/a	true	Intel® RDT monitoring technologies are supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMON =true
	RDTL3CA	n/a	true	Intel® RDT L3 Cache Allocation Technology is supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTL3CA= true
	RDTL2CA	n/a	true	Intel® RDT L2 Cache Allocation Technology is supported. Example: node.alpha.kubernetes-incubator.io/nfd-rdt-RDTL2CA= true
selinux	enabled	n/a	true	SELinux enforcing has been turned on in the Linux kernel. Example: node.alpha.kubernetes-incubator.io/nfd-selinux-enabled =true
storage	nonrotationaldisk	n/a	true	Non-rotational block device(s), like an SSD, is present. Example: node.alpha.kubernetes-incubator.io/nfd-storage-nonrotationaldisk = true

3 Deployment

3.1 Deployment as a DaemonSet

The preferred way to deploy NFD is to run it as a Kubernetes DaemonSet. This ensures that all nodes of the cluster run NFD, and new nodes get automatically labeled as soon as they become schedulable in the cluster. As a DaemonSet, NFD runs in the background, re-labeling nodes every 60 seconds (by default) so that any changes in the node capabilities are detected.

In its GitHub repository, NFD provides template specs that can be used for deployment:

```
$ kubectl create -f \
  https://raw.githubusercontent.com/kubernetes-incubator/node-feature-
discovery/master/rbac.yaml
$ kubectl create -f \
  https://raw.githubusercontent.com/kubernetes-incubator/node-feature-
discovery/master/node-feature-discovery-daemonset.yaml.template
```

This deploys the latest release of NFD, with the default configuration in the default Kubernetes namespace.

You can verify that NFD is running as expected by running:

```
$ kubectl  get  ds/node-feature-discovery
```

The output is similar to:

```
NAME                     DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
node-feature-discovery   5         5         5                    5           <none>          2m
```

Also, you can check the labels created by NFD:

```
$ kubectl  label  node  --list  -all

Listing  labels  for Node./node2:
 beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
 kubernetes.io/hostname=node2
 node.alpha.kubernetes-incubator.io/nfd-cpuid-AESNI=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-CLMUL=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-CMOV=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-CX16=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-ERMS=true
 node.alpha.kubernetes-incubator.io/nfd-cpuid-F16C=true
node.alpha.kubernetes-incubator.io/nfd-cpuid-MMX=true
```

3.2 Deployment as a Job

An alternative deployment mechanism is to run NFD as a one-shot Kubernetes Job. This may be a useful in a static cluster where no hardware changes are expected and the number of running pods need to be minimized. The NFD repository contains an example template and a deployment script that demonstrates this. You need to clone the repository in order to run this:

```
$ git  clone  https://github.com/kubernetes-incubator/node-feature-    discovery
&& cd node-feature-discovery
$  ./label-nodes.sh
```

The script launches as many instances of NFD as there are nodes in the Ready state in the cluster. However, this approach is not guaranteed to correctly run NFD on every node in all situations. For example, if some node is tainted NoSchedule or fails to start a job for some other reason, then NFD may not run correctly. For these reasons, we recommend that you use a DaemonSet deployment.

3.3 Deploying Custom-Built Version

Sometimes it may be desirable to run a self-built version of NFD, for example to try out an unreleased version. You must have installed Docker* and make to run the build. Follow the steps below:

1. Clone NFD source code:
   ```
   $ git clone https://github.com/kubernetes-incubator/node-feature-
   discovery
   $ cd node-feature-discovery
   ```
2. Next, run make to build the Docker image: `$ make`
3. Take a note of the NFD image hash built in the previous step, tag it and push the NFD image to your Docker registry, available for your Kubernetes cluster:
   ```
   $ docker tag <image hash> <docker registry>/<image name>
   $ docker push <docker registry>/<image name>
   ```
4. Edit node-feature-discovery-daemonset.yaml.template and change it to use your custom-built container image:
   ```
   ...
   Image: <docker  registry>/<image  name>
   ```

5. Deploy NFD:
```
$ kubectl apply -f node-feature-discovery-daemonset.yaml.template
```

4 Using Labels to Schedule Pods

Deploying NFD in a Kubernetes cluster labels all schedulable nodes according to the underlying platform features. These labels can be used for placing hard or soft constraints on where specific pods should be run. This section describes the two mechanisms to achieve this: **nodeSelector** and **nodeAffinity.**

4.1 nodeSelector

nodeSelector is a simple and limited mechanism to specify hard requirements on which node a pod should be run. nodeSelector contains a list of key-value pairs presenting required node labels and their values. A node must fulfill each of the requirements, that is, it must have each of the indicated label-value pairs in order for the pod to be able to be scheduled there.

The example below shows a pod specification requiring to be run on a node with SR-IOV capability:

```
apiVersion:  v1  kind: Pod
metadata:
  name: node-selector-example   spec:
  nodeSelector:
    node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable:"true"
  containers:
  - name:  nginx   image:
    nginx
```

4.2 nodeAffinity

nodeAffinity provides a much more expressive way to specify constraints on which nodes a pod should be run. It provides a range of different operators to use for matching label values (not just "equal to") and allows the specification of both hard and soft requirements (i.e., preferences).

The example below presents a pod specification where a pod is required to be run on a host with kernel version greater than 4.14, with a preference for Intel® Turbo Boost Technology being disabled (demonstrating soft anti-affinity).

```
apiVersion: v1 kind: Pod metadata:
name: node-affinity-example spec:
affinity: nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms:
- matchExpressions:
- key: node.alpha.kubernetes-incubator.io/nfd-kernel-
version.major
operator: Gt values: ["3"]
- key: node.alpha.kubernetes-incubator.io/nfd-kernel-
version.minor
        operator:  Gt   values:
        ["14"]
preference: matchExpressions:
- key: node.alpha.kubernetes-incubator.io/nfd-pstate-
turbo
operator: NotIn
values: ["true"] containers:
- name: nginx
image: nginx
```

5 Runtime Configuration

The template deployment specifications provided as part of the NFD source code repository (see Section 3 Deployment) specifies a default configuration that should be usable as-is for most users. However, NFD provides ways to alter its behavior through command line options and a configuration file.

5.1 Command Line Options

NFD has multiple command line options that can be used for tasks such as altering the set of labels to be advertised, for example. Specify the desired command line options under the Args keyword in the NFD pod specification.

Available command line options are listed in the following table.

Table 4. Command Line Options

Option	Description
--sources=<sources>	Comma-separated list of enabled feature sources. Can be used to limit detected features to a limited set of features. By default, all sources are enabled.
--label-whitelist=<pattern>	Regular expression to filter published label names. Empty by default, that is, no whitelist filter is enabled, and all labels are published.
--oneshot	Label once and exit after that. This is used in the one-shot Job configuration.
--sleep-interval=<time>	Interval of re-labeling. Specified using numbers and units ("s", "m", "h"), for example: "1m30s". Non-positive value implies no re-labeling (that is, infinite sleep). Does not have any effect if --oneshot is specified. Default interval is 60s.
--no-publish	Do not publish any labels. Useful for testing.
--config=<path>	NFD configuration file to read. Can be used to specify a custom location for the configuration file.
--options=<config>	Specify configuration options from command line, specified in the same format as in the configuration file (i.e. json or yaml). These options will override settings read from the configuration file. Useful for quickly testing configuration options and specifying a single configuration option.

5.2 Configuration File

Some aspects of NFD can be configured through an optional configuration file, which is located by default in `/etc/kubernetes/node-feature-discovery/node- feature-discovery.conf`. A custom location can be specified using the `--config` command line option. The configuration file must be available inside the NFD pod, and thus, Volumes and VolumeMounts are needed to make it available for NFD. The preferred method is to use a ConfigMap.

The following steps provide an example of creating and deploying a configuration map, using the example configuration from the NFD source code repository as a template.

1. Use the example configuration as a base for your customized configuration. `$ cp node-feature-discovery.conf.example node-feature- discovery.conf`

```
$ vim node-feature-discovery.conf # edit the configuration
```

2. Create a Kubernetes ConfigMap object from your configuration file.

```
$ kubectl create configmap node-feature-discovery-config --from-
file=node-feature-discovery.conf
```

3. Configure Volumes and VolumeMounts in the NFD pod spec: Note: Only the relevant code snippets are shown below.

```
     ...             containers:
         - volumeMounts:
             - name: node-feature-discovery-config
               mountPath: "/etc/kubernetes/node-feature-discovery/"    ...
       volumes:
         - name: node-feature-discovery-config   configMap:
             name: node-feature-discovery-config   ...
```

4. NFD will read your custom configuration file.

You could also use other types of volumes, of course. For example, hostPath could be used for local node-specific configurations.

The example configuration in the NFD source code repository is used as a configuration in the NFD container image. Thus, by directly editing the example configuration, you can alter the default configuration in custom-built images.

Configuration options can also be specified via the `--options` command line flag, in which case no mounts need to be used. This is mostly recommended for quickly testing configuration options and possibly specifying single options without the need to use ConfigMap. For example, a snippet from NFD DaemonSet specification:

```
... containers:

- args:

- '--options={"sources": { "pci": { "deviceClassWhitelist": ["12"] } } }'

- '--sleep-interval=60s' ...
```

Currently, the only available configuration options are related to the PCI feature source.

6 Summary

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

This document describes the usage and benefits of Node Feature Discovery in a Kubernetes deployment, including:

- Deployment of NFD
- Description of platform features discovered by NFD
- Using NFD labels for optimizing workload placement
- Runtime configuration of NFD

For more information on what Intel is doing with containers, see https://networkbuilders.intel.com/intel-technologies/container-experience-kits.

0222/DN/WIPRO/PDF 606833-003US