

# Node Feature Discovery

**Application Note**

---

*December 2018*



You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: <http://www.intel.com/design/literature.htm>

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at <http://www.intel.com/> or from the OEM or retailer.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2020, Intel Corporation. All rights reserved.



## Contents

---

<b>1.0</b>	<b>Introduction</b> .....	<b>5</b>
<b>2.0</b>	<b>Overview</b> .....	<b>6</b>
<b>3.0</b>	<b>How NFD Works</b> .....	<b>9</b>
<b>4.0</b>	<b>Feature Labels</b> .....	<b>10</b>
<b>5.0</b>	<b>Deployment</b> .....	<b>12</b>
5.1	Deployment as a DaemonSet.....	12
5.2	Deployment as a Job.....	13
5.3	Deploying Custom-Built Version.....	13
<b>6.0</b>	<b>Using Labels to Schedule Pods</b> .....	<b>14</b>
6.1	nodeSelector.....	14
6.2	nodeAffinity.....	14
<b>7.0</b>	<b>Runtime Configuration</b> .....	<b>16</b>
7.1	Command Line Options.....	16
7.2	Configuration File.....	16
<b>8.0</b>	<b>Summary</b> .....	<b>18</b>
<b>Appendix A Terminology and References</b> .....		<b>19</b>

## Figures

Figure 1.	Node Feature List.....	8
Figure 2.	Node Feature Discovery in Kubernetes.....	9

## Tables

Table 1.	Feature Labels.....	10
Table 2.	Command Line Options.....	16
Table 3.	Terminology.....	19
Table 4.	References.....	19



## *Revision History*

---

Date	Revision	Description
December 2018	001	Initial release.



## 1.0 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 in order to facilitate improved workload placement based on platform capabilities.

Node Feature Discovery is part of the Enhanced Platform Awareness (EPA) suite which represents a methodology and a set of changes in Kubernetes targeting intelligent configuration and capacity consumption of platform capabilities. Through increased awareness and orchestration of system resources and capabilities, EPA delivers improved application performance and determinism.

NFD is an open source Kubernetes community project. The software is available at: <https://github.com/kubernetes-sigs/node-feature-discovery>

This document is part of the Container Experience Kit for Enhanced Platform Awareness (EPA). Container Experience Kits are collections of user guides, application notes, feature briefs, and other collateral that provide a library of best-practice documents for engineers who are developing container-based applications. Other documents in the EPA Container Experience Kit can be found at: <https://networkbuilders.intel.com/network-technologies/container-experience-kits>

**Note:** This document does not describe how to set up a Kubernetes cluster. We recommend that you perform those steps as a prerequisite. For more setup and installation guidelines of a complete system, refer to the *Deploying Kubernetes and Container Bare Metal Platform for NFV Use Cases with Intel® Xeon® Scalable Processors User Guide* listed in [Table 4](#).

The relevant documents include:

Document Title	Document Type
Enhanced Platform Awareness in Kubernetes	Feature Brief
Enhanced Platform Awareness in Kubernetes	Application Note
Enabling New Features with Kubernetes for NFV	White Paper
Enhanced Platform Awareness in Kubernetes	Performance Benchmark Report



## 2.0 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). Certain workloads, such as machine learning, may gain a significant performance improvement from these extensions (e.g. 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® Turbo Boost Technology:** Intel® Turbo Boost Technology accelerates processor performance for peak loads, dynamically overclocking processor cores if



they are operating within the power, current, and temperature limits of the processor. This can provide significant performance benefits for CPU-bound workloads. On the other hand, some workloads behave better when this technology has been disabled. 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. 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.
- **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, allowing optimized scheduling of workloads dependent on certain PCI devices.

NFD is under active development, and in 2018 it has evolved significantly, gaining new functionality, including the discovery of multiple new features, as shown in [Figure 1](#).



Figure 1. Node Feature List

X86 CPUID Features (Partial List)	
Feature Name	Description
ADX	Multi-Precision Add-Carry Instruction Extensions (ADX)
AESNI	Advanced Encryption Standard (AES) New Instructions (AES-NI)
AVX	Advanced Vector Extensions (AVX)
AVX2	Advanced Vector Extensions 2 (AVX2)
BMI1	Bit Manipulation Instruction Set 1 (BMI)
BMI2	Bit Manipulation Instruction Set 2 (BMI2)
SSE4.1	Streaming SIMD Extensions 4.1 (SSE4.1)
SSE4.2	Streaming SIMD Extensions 4.2 (SSE4.2)

RDT (Intel resource Director Technology) Features	
Feature Name	Description
RDTMON	Intel RDT Monitoring Technology
RDTCMT	Intel Cache Monitoring (CMT)
RDTMBM	Intel Memory Bandwidth Monitoring (MBM)
RDTL3CA	Intel L3 Cache Allocation Technology
RDTL2CA	Intel L2 Cache Allocation Technology
RDTMBA	Intel Memory Bandwidth Allocation (MBA) Technology

Selinux Features	
Feature Name	Description
selinux	selinux is enabled on the node

Storage Features	
Feature Name	Description
nonrotationdisk	Non-rotational disk, like SSD, is present in the node

IOMMU Features	
Feature Name	Description
enabled	IOMMU is present and enabled in the kernel

Arm64 CPUID Features (Partial List)	
Feature Name	Description
AES	Announcing the Advanced Encryption Standard
EVSTRM	Event Stream Frequency Features
FPHP	Half Precision(16bit) Floating Point Data Processing Instructions
AISMDHP	Half Precision(16bit) Asimd Data Processing Instructions
ATOMICS	Atomic Instructions to the A64Manipulation Instruction Set 1 (BMI)
AISMRDM	Support for Rounding Double Multiply Add/Subtract
PMULL	Optional Cryptographic and CRC32 Instructions
JSCVT	Perform Conversion to Match Javascript

Memory Features	
Feature Name	Description
numa	Multiple memory nodes i.e. NUMA architecture detected

Network Features		
Feature Name	Attribute	Description
SRIOV	capable	Single Root Input/Output Virtualization (SR-IOV) enabled Network Interface Card(s) present
	configured	SR-IOV virtual functions have been configured

Kernel Features		
Feature Name	Attribute	Description
version	full	Full kernel version as reported by /proc/sys/kernel/osrelease (e.g. '4.5.6-7-g123abcde')
	major	First component of the kernel version (e.g. '4')
	minor	Second component of the kernel version (e.g. '5')
	revision	Third component of the kernel version (e.g. '6')

PCI Features		
Feature Name	Attribute	Description
<device label>	present	PCI device is detected

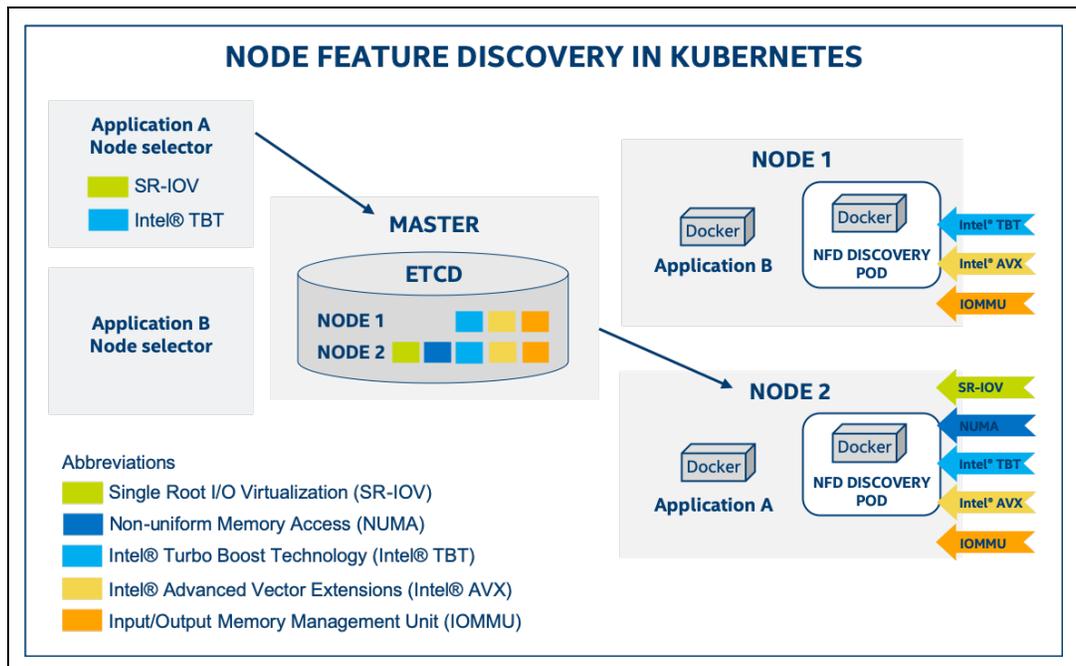


### 3.0 How NFD Works

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 2](#) 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 2](#) 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 [Figure 2](#), this is Node 1. Application B has no special capability requests, therefore it can be placed on either node. In [Figure 2](#), it has been placed on Node 2.

**Figure 2. Node Feature Discovery in Kubernetes**





## 4.0 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 1. Feature Labels**

Source	Feature	Attribute	Possible values	Description
cpuid	<cpuid feature name>	<i>n/a</i>	true	All CPU features returned by the CPUID instruction. Examples: <a href="#">node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX = true</a> <a href="#">node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512F = true</a> <a href="#">node.alpha.kubernetes-incubator.io/nfd-cpuid-SHA = true</a>
kernel	version	full	<version string>	Full kernel version. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-kernel-version.full = 4.5.6-7-g123abcde</a>
		major	<version number>	First component of the kernel version. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-kernel-version.major = 4</a>
		minor	<version number>	Second component of the kernel version. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-kernel-version.minor = 5</a>



Source	Feature	Attribute	Possible values	Description
		revision	<version number>	Third component of the kernel version. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-kernel-version.minor = 6</a>
iommu	enabled	n/a	true	An IOMMU is present and enabled in the kernel. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-iommu-enabled = true</a>
memory	numa	n/a	true	NUMA topology detected. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-memory-numa = true</a>
network	sriov	capable	true	SR-IOV capable Network Interface Card(s) present. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-network.sriov.capable = true</a>
		configured	true	SR-IOV Virtual Functions have been configured. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-network.sriov.configured = true</a>
pci	<device label>	present	true	Presence of PCI device is detected. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-pci-1200_8086.present=true</a>
		<p><b>NOTE:</b> &lt;device label&gt; is composed of raw PCI IDs, separated by underscores. The set of fields is configurable, valid fields being <b>class</b>, <b>vendor</b>, <b>device</b>, <b>subsystem_vendor</b> and <b>subsystem_device</b> (defaults are <b>class</b> and <b>vendor</b>). Also the set of PCI device classes that the feature source detects is configurable. By default, device classes <b>03(h)</b>, <b>0b40(h)</b> and <b>12(h)</b>, i.e. GPUs, co-processors, and accelerator cards, are detected. See the <a href="#">Runtime Configuration</a> section for more details about NFD configuration options.</p>		
rdt	RDTCMT	n/a	true	Intel® RDT Cache Monitoring Technology is supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTCMT = true</a>
	RDTMBM	n/a	true	Intel® RDT Memory Bandwidth Monitoring is supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBM = true</a>
	RDTMBA	n/a	true	Intel® RDT Memory Bandwidth Allocation is supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMBA = true</a>
	RDTMON	n/a	true	Intel® RDT monitoring technologies are supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMON = true</a>
	RDTL3CA	n/a	true	Intel® RDT L3 Cache Allocation Technology is supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTL3CA= true</a>
	RDTL2CA	n/a	true	Intel® RDT L2 Cache Allocation Technology is supported. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-rdt-RDTL2CA= true</a>
selinux	enabled	n/a	true	SELinux enforcing has been turned on in the Linux kernel. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-selinux-enabled = true</a>
storage	nonrotationaldisk	n/a	true	Non-rotational block device(s), like an SSD, is present. Example: <a href="#">node.alpha.kubernetes-incubator.io/nfd-storage-nonrotationaldisk = true</a>



## 5.0 Deployment

---

### 5.1 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 also, 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
...
```



## 5.2 Deployment as a Job

An alternative deployment mechanism is to run NFD as a one-shot Kubernetes Job. This may be 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.

## 5.3 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
```



## 6.0 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**.

### 6.1 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
```

### 6.2 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"]
        preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
                - key: node.alpha.kubernetes-incubator.io/nfd-pstate-
turbo
                  operator: NotIn
                  values: ["true"]
  containers:
    - name: nginx
      image: nginx
```



## 7.0 Runtime Configuration

The template deployment specs provided as part of the NFD source code repository (see the [Deployment](#) section) 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.

### 7.1 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 2. Command Line Options**

Option	Description
<code>--sources=&lt;sources&gt;</code>	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.
<code>--label-whitelist=&lt;pattern&gt;</code>	Regular expression to filter published label names. Empty by default, that is, no whitelist filter is enabled and all labels are published.
<code>--oneshot</code>	Label once and exit after that. This is used in the one-shot <b>Job</b> configuration.
<code>--sleep-interval=&lt;time&gt;</code>	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 <code>--oneshot</code> is specified. Default interval is 60s.
<code>--no-publish</code>	Do not publish any labels. Useful for testing.
<code>--config=&lt;path&gt;</code>	NFD configuration file to read. Can be used to specify a custom location for the configuration file.
<code>--options=&lt;config&gt;</code>	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.

### 7.2 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.



## 8.0 Summary

---

Together with other EPA 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/network-technologies/intel-container-experience-kits>



## Appendix A Terminology and References

**Table 3. Terminology**

Term	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

**Table 4. References**

Document	Document No./Location
Deploying Kubernetes and Container Bare Metal Platform for NFV Use Cases with Intel® Xeon® Scalable Processors	<a href="https://builders.intel.com/docs/networkbuilders/deploying-kubernetes-and-container-bare-metal-platform-for-nfv-use-cases-with-intel-xeon-scalable-processors-user-guide.pdf">https://builders.intel.com/docs/networkbuilders/deploying-kubernetes-and-container-bare-metal-platform-for-nfv-use-cases-with-intel-xeon-scalable-processors-user-guide.pdf</a>