



OpenShift Container Platform 4.12

특수 하드웨어 및 드라이버 활성화

OpenShift Container Platform의 하드웨어 활성화 관련 정보

OpenShift Container Platform 4.12 특수 하드웨어 및 드라이버 활성화

OpenShift Container Platform의 하드웨어 활성화 관련 정보

법적 공지

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

이 문서에서는 OpenShift Container Platform의 하드웨어 활성화에 대한 개요를 설명합니다.

차례

1장. 특수 하드웨어 및 드라이버 활성화 정보	3
2장. 드라이버 툴킷	4
2.1. 드라이버 툴킷 정보	4
2.2. DRIVER TOOLKIT 컨테이너 이미지 가져오기	5
2.3. DRIVER TOOLKIT 사용	6
2.4. 추가 리소스	10
3장. NODE FEATURE DISCOVERY OPERATOR	11
3.1. NODE FEATURE DISCOVERY OPERATOR 정보	11
3.2. NODE FEATURE DISCOVERY OPERATOR 설치	11
3.3. NODE FEATURE DISCOVERY OPERATOR 사용	13
3.4. NODE FEATURE DISCOVERY OPERATOR 설정	16
3.5. NFD 토폴로지 업데이트 프로그램 사용	21
4장. KERNEL MODULE MANAGEMENT OPERATOR	26
4.1. KERNEL MODULE MANAGEMENT OPERATOR 정보	26
4.2. KERNEL MODULE MANAGEMENT OPERATOR 설치	26
4.3. 커널 모듈 관리(KMM)에서 서명 사용	30
4.4. SECUREBOOT를 위한 키 추가	31
4.5. 사전 빌드된 드라이버 컨테이너 서명	32
4.6. MODULELOADER 컨테이너 이미지 빌드 및 서명	33
4.7. 디버깅 및 문제 해결	36

1장. 특수 하드웨어 및 드라이버 활성화 정보

Driver Toolkit(DTK)은 드라이버 컨테이너를 빌드하는 기본 이미지로 사용되는 OpenShift Container Platform 페이로드의 컨테이너 이미지입니다. Driver Toolkit 이미지에는 커널 모듈을 빌드하거나 설치하는 데 일반적으로 필요한 커널 패키지와 드라이버 컨테이너에 필요한 몇 가지 도구가 포함되어 있습니다. 이러한 패키지의 버전은 해당 OpenShift Container Platform 릴리스의 RHCOS 노드에서 실행되는 커널 버전과 일치합니다.

드라이버 컨테이너는 `:op-system-first:`와 같은 컨테이너 운영 체제에서 트리 외부 커널 모듈 및 드라이버를 빌드하고 배포하는 데 사용되는 컨테이너 이미지입니다. 커널 모듈과 드라이버는 운영 체제 커널에서 높은 수준의 권한으로 실행되는 소프트웨어 라이브러리입니다. 커널 기능을 확장하거나 새 장치를 제어하는 데 필요한 하드웨어별 코드를 제공합니다. 예를 들면 필드 프로그래밍 가능 게이트 어레이(FPGA) 또는 그래픽 처리 장치(GPU)와 같은 하드웨어 장치 및 모두 클라이언트 시스템에 커널 모듈이 필요한 소프트웨어 정의 스토리지 솔루션이 있습니다. 드라이버 컨테이너는 OpenShift Container Platform 배포에서 이러한 기술을 활성화하는 데 사용되는 소프트웨어 스택의 첫 번째 계층입니다.

2장. 드라이버 툴킷

Driver Toolkit과 OpenShift Container Platform 배포에서 특수 소프트웨어 및 하드웨어 장치를 활성화하기 위한 드라이버 컨테이너의 기본 이미지로 사용할 수 있는 방법에 대해 알아보십시오.

2.1. 드라이버 툴킷 정보

배경

Driver Toolkit은 드라이버 컨테이너를 빌드할 수 있는 기본 이미지로 사용되는 OpenShift Container Platform 페이로드의 컨테이너 이미지입니다. Driver Toolkit 이미지는 커널 모듈을 빌드하거나 설치하는 데 일반적으로 필요한 커널 패키지와 드라이버 컨테이너에 필요한 몇 가지 툴이 포함되어 있습니다. 이러한 패키지의 버전은 해당 OpenShift Container Platform 릴리스의 RHCOS(Red Hat Enterprise Linux CoreOS) 노드에서 실행되는 커널 버전과 동일합니다.

드라이버 컨테이너는 RHCOS와 같은 컨테이너 운영 체제에서 트리 외부 커널 모듈 및 드라이버를 빌드하고 배포하는 데 사용되는 컨테이너 이미지입니다. 커널 모듈과 드라이버는 운영 체제 커널에서 높은 수준의 권한으로 실행되는 소프트웨어 라이브러리입니다. 커널 기능을 확장하거나 새 장치를 제어하는 데 필요한 하드웨어별 코드를 제공합니다. 예를 들어 Field Programmable Gate Arrays(예: Field Programmable Gate Arrays) 또는 GPU와 같은 하드웨어 장치, Lustre 병렬 파일 시스템과 같은 소프트웨어 정의 스토리지(SDS) 솔루션은 클라이언트 시스템에 커널 모듈이 필요합니다. 드라이버 컨테이너는 Kubernetes에서 이러한 기술을 활성화하는 데 사용되는 소프트웨어 스택의 첫 번째 계층입니다.

Driver Toolkit의 커널 패키지 목록에는 다음과 같은 종속성이 포함되어 있습니다.

- **kernel-core**
- **kernel-devel**
- **kernel-headers**
- **kernel-modules**
- **kernel-modules-extra**

또한 Driver Toolkit에는 해당 실시간 커널 패키지도 포함되어 있습니다.

- **kernel-rt-core**
- **kernel-rt-devel**
- **kernel-rt-modules**
- **kernel-rt-modules-extra**

또한 Driver Toolkit에는 다음을 포함하여 커널 모듈을 빌드하고 설치하는 데 일반적으로 필요한 여러 도구가 있습니다.

- **elfutils-libelf-devel**
- **kmod**
- **binutils-kabi-dw**
- **kernel-abi-whitelists**
- 위의 종속 항목

목적

Driver Toolkit이 존재하기 전에 사용자는 [권한이 부여된 빌드](#) 를 사용하거나 호스트 **machine-os-content** 의 커널 RPM에서 설치하여 OpenShift Container Platform의 Pod에 커널 패키지를 설치합니다. Driver Toolkit은 인타이틀먼트 단계를 제거하여 프로세스를 간소화하고 Pod에서 machine-os-content에 액세스 하는 권한 있는 작업을 피할 수 있습니다. Driver Toolkit은 사전 릴리스된 OpenShift Container Platform 버전에 액세스할 수 있는 파트너가 향후 OpenShift Container Platform 릴리스를 위한 하드웨어 장치용 드라이버 컨테이너를 사전 구축하는 데 사용할 수도 있습니다.

Driver Toolkit은 현재 OperatorHub에서 커뮤니티 Operator로 사용할 수 있는 KMM(커널 모듈 관리)에서도 사용됩니다. KMM은 out-of-tree 및 타사 커널 드라이버 및 기본 운영 체제에 대한 지원 소프트웨어를 지원합니다. KMM용 모듈을 생성하여 드라이버 컨테이너를 빌드하고 배포할 수 있으며 장치 플러그인 또는 메트릭과 같은 지원 소프트웨어를 생성할 수 있습니다. 모듈에는 Driver Toolkit을 기반으로 드라이버 컨테이너를 빌드하는 빌드 구성이 포함될 수 있습니다. 또는 KMM은 사전 빌드된 드라이버 컨테이너를 배포할 수 있습니다.

2.2. DRIVER TOOLKIT 컨테이너 이미지 가져오기

driver-toolkit 이미지는 [Red Hat Ecosystem Catalog](#)의 [컨테이너 이미지 섹션](#)과 OpenShift Container Platform 릴리스 페이지로드에서 사용할 수 있습니다. OpenShift Container Platform의 최신 마이너 릴리스에 해당하는 이미지는 카탈로그의 버전 번호로 태그가 지정됩니다. 특정 릴리스의 이미지 URL은 **oc adm** CLI 명령을 사용하여 찾을 수 있습니다.

2.2.1. registry.redhat.io에서 Driver Toolkit 컨테이너 이미지 가져오기

registry.redhat.io 에서 **podman** 또는 OpenShift Container Platform을 사용하여 **driver-toolkit** 이미지를 가져오는 방법은 [Red Hat Ecosystem Catalog](#) 에서 확인할 수 있습니다. 최신 마이너 릴리스의 driver-toolkit 이미지는 **registry.redhat.io /openshift4/driver-toolkit-rhel8:v4.12** 의 마이너 릴리스 버전으로 태그가 지정됩니다.

2.2.2. 페이지로드에서 Driver Toolkit 이미지 URL 검색

사전 요구 사항

- [Red Hat OpenShift Cluster Manager](#)에서 이미지 풀 시크릿을 가져왔습니다 .
- OpenShift CLI(**oc**)를 설치합니다.

절차

1. **oc adm** 명령을 사용하여 특정 릴리스에 해당하는 **driver-toolkit** 의 이미지 URL을 추출합니다.

- x86 이미지의 경우 다음 명령을 입력합니다.

```
$ oc adm release info quay.io/openshift-release-dev/ocp-release:4.12.z-x86_64 --image-for=driver-toolkit
```

- ARM 이미지의 경우 다음 명령을 입력합니다.

```
$ oc adm release info quay.io/openshift-release-dev/ocp-release:4.12.z-aarch64 --image-for=driver-toolkit
```

출력 예

```
quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:0fd84aee79606178b6561ac71f8540f404d518ae5deff45f6d6ac8f02636c7f4
```

2. OpenShift Container Platform을 설치하는 데 필요한 풀 시크릿과 같은 유효한 풀 시크릿을 사용하여 이 이미지를 가져옵니다.

```
$ podman pull --authfile=path/to/pullsecret.json quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:<SHA>
```

2.3. DRIVER TOOLKIT 사용

예를 들어 Driver Toolkit은 **simple-kmod** 라는 매우 간단한 커널 모듈을 빌드하기 위한 기본 이미지로 사용할 수 있습니다.



참고

Driver Toolkit에는 커널 모듈에 서명하는 데 필요한 종속성, **openssl, mokutil, keyutils** 가 포함되어 있습니다. 그러나 이 예에서는 **simple-kmod** 커널 모듈이 서명되지 않으므로 **Secure Boot** 가 활성화된 시스템에 로드할 수 없습니다.

2.3.1. 클러스터에서 **simple-kmod** 드라이버 컨테이너를 빌드하고 실행합니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터가 있어야 합니다.
- Image Registry Operator 상태를 클러스터의 **Managed**로 설정합니다.
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 OpenShift CLI에 로그인했습니다.

절차

네임스페이스를 생성합니다. 예를 들면 다음과 같습니다.

```
$ oc new-project simple-kmod-demo
```

1. YAML은 **simple-kmod** 드라이버 컨테이너 이미지를 저장하기 위한 **ImageStream**과 컨테이너 빌드를 위한 **BuildConfig**를 정의합니다. 이 YAML을 **0000-buildconfig.yaml.template**로 저장합니다.

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  labels:
    app: simple-kmod-driver-container
    name: simple-kmod-driver-container
    namespace: simple-kmod-demo
spec: {}
---
apiVersion: build.openshift.io/v1
kind: BuildConfig
```

```

metadata:
  labels:
    app: simple-kmod-driver-build
    name: simple-kmod-driver-build
    namespace: simple-kmod-demo
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  runPolicy: "Serial"
  triggers:
    - type: "ConfigChange"
    - type: "ImageChange"
  source:
    dockerfile: |
      ARG DTK
      FROM ${DTK} as builder

      ARG KVER

      WORKDIR /build/

      RUN git clone https://github.com/openshift-psap/simple-kmod.git

      WORKDIR /build/simple-kmod

      RUN make all install KVER=${KVER}

      FROM registry.redhat.io/ubi8/ubi-minimal

      ARG KVER

      # Required for installing `modprobe`
      RUN microdnf install kmod

      COPY --from=builder /lib/modules/${KVER}/simple-kmod.ko /lib/modules/${KVER}/
      COPY --from=builder /lib/modules/${KVER}/simple-procfs-kmod.ko
/lib/modules/${KVER}/
      RUN depmod ${KVER}
  strategy:
    dockerStrategy:
      buildArgs:
        - name: KMODVER
          value: DEMO
          # $ oc adm release info quay.io/openshift-release-dev/ocp-release:<cluster version>-
x86_64 --image-for=driver-toolkit
        - name: DTK
          value: quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:34864ccd2f4b6e385705a730864c04a40908e57acede44457a783d739e377cae
        - name: KVER
          value: 4.18.0-372.26.1.el8_6.x86_64
    output:
      to:
        kind: ImageStreamTag
        name: simple-kmod-driver-container:demo

```

2. "DRIVER_TOOLKIT_IMAGE" 대신 실행 중인 OpenShift Container Platform 버전에 대해 올바른 드라이버 툴킷 이미지를 다음 명령으로 대체하십시오.

```
$ OCP_VERSION=$(oc get clusterversion/version -ojsonpath={.status.desired.version})
```

```
$ DRIVER_TOOLKIT_IMAGE=$(oc adm release info $OCP_VERSION --image-for=driver-toolkit)
```

```
$ sed "s#DRIVER_TOOLKIT_IMAGE#{$DRIVER_TOOLKIT_IMAGE}#" 0000-buildconfig.yaml.template > 0000-buildconfig.yaml
```

3. 이미지 스트림 및 빌드 구성을 만듭니다.

```
$ oc create -f 0000-buildconfig.yaml
```

4. 빌더 Pod가 성공적으로 완료되면 드라이버 컨테이너 이미지를 **DaemonSet**으로 배포합니다.

- a. 호스트에서 커널 모듈을 로드하려면 드라이버 컨테이너를 권한 있는 보안 컨텍스트로 실행해야 합니다. 다음 YAML 파일에는 RBAC 규칙과 드라이버 컨테이너 실행을 위한 **DaemonSet**이 포함되어 있습니다. 이 YAML을 **1000-drivercontainer.yaml**로 저장합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: simple-kmod-driver-container
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: simple-kmod-driver-container
rules:
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  verbs:
  - use
  resourceNames:
  - privileged
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: simple-kmod-driver-container
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: simple-kmod-driver-container
subjects:
- kind: ServiceAccount
  name: simple-kmod-driver-container
userNames:
- system:serviceaccount:simple-kmod-demo:simple-kmod-driver-container
---
```

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: simple-kmod-driver-container
spec:
  selector:
    matchLabels:
      app: simple-kmod-driver-container
  template:
    metadata:
      labels:
        app: simple-kmod-driver-container
    spec:
      serviceAccount: simple-kmod-driver-container
      serviceAccountName: simple-kmod-driver-container
      containers:
        - image: image-registry.openshift-image-registry.svc:5000/simple-kmod-
          demo/simple-kmod-driver-container:demo
          name: simple-kmod-driver-container
          imagePullPolicy: Always
          command: [sleep, infinity]
          lifecycle:
            postStart:
              exec:
                command: ["modprobe", "-v", "-a", "simple-kmod", "simple-procfs-kmod"]
            preStop:
              exec:
                command: ["modprobe", "-r", "-a", "simple-kmod", "simple-procfs-kmod"]
          securityContext:
            privileged: true
      nodeSelector:
        node-role.kubernetes.io/worker: ""

```

- b. RBAC 규칙 및 데몬 세트를 생성합니다.

```
$ oc create -f 1000-drivercontainer.yaml
```

5. 작업자 노드에서 pod가 실행된 후 **lsmod**가 있는 호스트 시스템에 **simple_kmod** 커널 모듈이 제대로 로드되었는지 확인합니다.

- a. pod가 실행 중인지 확인합니다.

```
$ oc get pod -n simple-kmod-demo
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
simple-kmod-driver-build-1-build	0/1	Completed	0	6m
simple-kmod-driver-container-b22fd	1/1	Running	0	40s
simple-kmod-driver-container-jz9vn	1/1	Running	0	40s
simple-kmod-driver-container-p45cc	1/1	Running	0	40s

- b. 드라이버 컨테이너 Pod에서 **lsmod** 명령을 실행합니다.

```
$ oc exec -it pod/simple-kmod-driver-container-p45cc -- lsmod | grep simple
```

출력 예

```
simple_procfs_kmod 16384 0
simple_kmod         16384 0
```

2.4. 추가 리소스

- 클러스터의 레지스트리 스토리지 설정에 대한 자세한 내용은 [OpenShift Container Platform의 Image Registry Operator](#) 를 참조하십시오.

3장. NODE FEATURE DISCOVERY OPERATOR

Node Feature Discovery (NFD) Operator 및 이를 사용하여 하드웨어 기능과 시스템 구성을 감지하기 위한 Kubernetes 애드온 기능인 NFD(Node Feature Discovery)을 오케스트레이션하여 노드 수준 정보를 공개하는 방법을 설명합니다.

3.1. NODE FEATURE DISCOVERY OPERATOR 정보

NFD(Node Feature Discovery Operator)는 하드웨어 관련 정보로 노드에 레이블을 지정하여 OpenShift Container Platform 클러스터에서 하드웨어 기능 및 구성 검색을 관리합니다. NFD는 PCI 카드, 커널, 운영 체제 버전과 같은 노드별 속성을 사용하여 호스트에 레이블을 지정합니다.

NFD Operator는 "Node Feature Discovery"을 검색하여 Operator Hub에서 확인할 수 있습니다.

3.2. NODE FEATURE DISCOVERY OPERATOR 설치

NFD(Node Feature Discovery) Operator는 NFD 데몬 세트를 실행하는 데 필요한 모든 리소스를 오케스트레이션합니다. 클러스터 관리자는 OpenShift Container Platform CLI 또는 웹 콘솔을 사용하여 NFD Operator를 설치할 수 있습니다.

3.2.1. CLI를 사용하여 NFD Operator 설치

클러스터 관리자는 CLI를 사용하여 NFD Operator를 설치할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

절차

1. NFD Operator의 네임스페이스를 생성합니다.

- a. **openshift-nfd** 네임스페이스를 정의하는 다음 **Namespace** CR(사용자 정의 리소스)을 생성하고 **nfd-namespace.yaml** 파일에 YAML을 저장합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-nfd
```

- b. 다음 명령을 실행하여 네임스페이스를 생성합니다.

```
$ oc create -f nfd-namespace.yaml
```

2. 다음 오브젝트를 생성하여 이전 단계에서 생성한 네임스페이스에 NFD Operator를 설치합니다.

- a. 다음 **OperatorGroup** CR을 생성하고 해당 YAML을 **nfd-operatorgroup.yaml** 파일에 저장합니다.

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  generateName: openshift-nfd-
  name: openshift-nfd
  namespace: openshift-nfd
spec:
  targetNamespaces:
    - openshift-nfd

```

- b. 다음 명령을 실행하여 **OperatorGroup** CR을 생성합니다.

```
$ oc create -f nfd-operatorgroup.yaml
```

- c. 다음 **Subscription** CR을 생성하고 해당 YAML을 **nfd-sub.yaml** 파일에 저장합니다.

서브스크립션의 예

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: nfd
  namespace: openshift-nfd
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: nfd
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

- d. 다음 명령을 실행하여 서브스크립션 오브젝트를 생성합니다.

```
$ oc create -f nfd-sub.yaml
```

- e. **openshift-nfd** 프로젝트로 변경합니다.

```
$ oc project openshift-nfd
```

검증

- Operator 배포가 완료되었는지 확인하려면 다음을 실행합니다.

```
$ oc get pods
```

출력 예

```

NAME                                READY STATUS RESTARTS AGE
nfd-controller-manager-7f86ccfb58-vgr4x 2/2   Running 0      10m

```

성공적인 배포에는 **Running** 상태가 표시됩니다.

3.2.2. 웹 콘솔을 사용하여 NFD Operator 설치

클러스터 관리자는 웹 콘솔을 사용하여 NFD Operator를 설치할 수 있습니다.

절차

1. OpenShift Container Platform 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.
2. 사용 가능한 Operator 목록에서 **Node Feature Discovery**를 선택한 다음 **설치**를 클릭합니다.
3. **Operator 설치** 페이지에서 **클러스터의 특정 네임스페이스**를 선택한 다음 **설치**를 클릭합니다. 네임스페이스가 생성되므로 생성할 필요가 없습니다.

검증

다음과 같이 NFD Operator가 설치되었는지 확인합니다.

1. **Operator** → **설치된 Operator** 페이지로 이동합니다.
2. **Node Feature Discovery**가 **openshift-nfd** 프로젝트에 **InstallSucceeded** 상태로 나열되어 있는지 확인합니다.



참고

설치 중에 Operator는 **실패** 상태를 표시할 수 있습니다. 나중에 **InstallSucceeded** 메시지와 함께 설치에 성공하면 이 **실패** 메시지를 무시할 수 있습니다.

문제 해결

Operator가 설치된 것으로 나타나지 않으면 다음과 같이 추가 문제 해결을 수행합니다.

1. **Operator** → **설치된 Operator** 페이지로 이동하고 **Operator** 서브스크립션 및 **설치 계획** 탭의 상태에 장애나 오류가 있는지 검사합니다.
2. **Workloads** → **Pod** 페이지로 이동하여 **openshift-nfd** 프로젝트에서 Pod 로그를 확인합니다.

3.3. NODE FEATURE DISCOVERY OPERATOR 사용

NFD(Node Feature Discovery) Operator는 **NodeFeatureDiscovery** CR을 확인하여 Node-Versionature-Discovery 데몬 세트를 실행하는 데 필요한 모든 리소스를 오케스트레이션합니다.

NodefeatureatureDiscovery CR을 기반으로 Operator는 원하는 네임스페이스에 피연산자(NFD) 구성 요소를 생성합니다. CR을 편집하여 다른 옵션 중에서 다른 **namespace**, **image**, **imagePullPolicy**, **nfd-worker-conf**를 선택할 수 있습니다.

클러스터 관리자는 OpenShift Container Platform CLI 또는 웹 콘솔을 사용하여 **NodeFeatureDiscovery** 인스턴스를 만들 수 있습니다.

3.3.1. CLI를 사용하여 NodeEnatureDiscovery 인스턴스 생성

클러스터 관리자는 CLI를 사용하여 **NodefeatureatureDiscovery** CR 인스턴스를 생성할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터
- OpenShift CLI(**oc**)를 설치합니다.

- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- NFD Operator를 설치합니다.

절차

1. 다음 **NodeEnatureDiscovery** 사용자 정의 리소스(CR)를 생성한 다음 YAML을 **NodefeatureatureDiscovery.yaml** 파일에 저장합니다.

```

apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: nfd-instance
  namespace: openshift-nfd
spec:
  instance: "" # instance is empty by default
  topologyupdater: false # False by default
  operand:
    image: registry.redhat.io/openshift4/ose-node-feature-discovery:v4.12
    imagePullPolicy: Always
  workerConfig:
    configData: |
      core:
        # labelWhiteList:
        # noPublish: false
        sleepInterval: 60s
        # sources: [all]
        # klog:
        # addDirHeader: false
        # alsologtostderr: false
        # logBacktraceAt:
        # logtostderr: true
        # skipHeaders: false
        # stderrthreshold: 2
        # v: 0
        # vmodule:
        ## NOTE: the following options are not dynamically run-time configurable
        ## and require a nfd-worker restart to take effect after being changed
        # logDir:
        # logFile:
        # logFileMaxSize: 1800
        # skipLogHeaders: false
      sources:
        cpu:
          cpuid:
            # NOTE: whitelist has priority over blacklist
            attributeBlacklist:
              - "BMI1"
              - "BMI2"
              - "CLMUL"
              - "CMOV"
              - "CX16"
              - "ERMS"
              - "F16C"
              - "HTT"
              - "LZCNT"

```

```

- "MMX"
- "MMXEXT"
- "NX"
- "POPCNT"
- "RDRAND"
- "RDSEED"
- "RDTSCP"
- "SGX"
- "SSE"
- "SSE2"
- "SSE3"
- "SSE4.1"
- "SSE4.2"
- "SSSE3"
attributeWhitelist:
kernel:
  kconfigFile: "/path/to/kconfig"
  configOpts:
    - "NO_HZ"
    - "X86"
    - "DMI"
pci:
  deviceClassWhitelist:
    - "0200"
    - "03"
    - "12"
  deviceLabelFields:
    - "class"
customConfig:
  configData: |
    - name: "more.kernel.features"
  matchOn:
    - loadedKMod: ["example_kmod3"]

```

NFD 작업을 사용자 지정하는 방법에 대한 자세한 내용은 [nfd-worker의 구성 파일 참조](#) 를 참조하십시오.

1. 다음 명령어를 실행하여 **NodeFeatureDiscovery** CR 인스턴스를 만듭니다.

```
$ oc create -f NodeFeatureDiscovery.yaml
```

검증

- 인스턴스가 생성되었는지 확인하려면 다음을 실행합니다.

```
$ oc get pods
```

출력 예

```

NAME                                READY STATUS RESTARTS AGE
nfd-controller-manager-7f86ccfb58-vgr4x 2/2   Running 0    11m
nfd-master-hcn64                        1/1   Running 0    60s
nfd-master-lnnxx                        1/1   Running 0    60s

```

nfd-master-mp6hr	1/1	Running	0	60s
nfd-worker-vgcz9	1/1	Running	0	60s
nfd-worker-xqbws	1/1	Running	0	60s

성공적인 배포에는 **Running** 상태가 표시됩니다.

3.3.2. 웹 콘솔을 사용하여 NodeEnatureDiscovery CR 만들기

절차

1. Operator → 설치된 Operator 페이지로 이동합니다.
2. Node Feature Discovery를 찾고 제공된 APIs아래에 있는 상자를 확인합니다.
3. 인스턴스 만들기를 클릭합니다.
4. NodefeatureatureDiscovery CR의 값을 편집합니다.
5. 생성을 클릭합니다.

3.4. NODE FEATURE DISCOVERY OPERATOR 설정

3.4.1. 코어

core 섹션에는 특정 기능 소스와 관련이 없는 일반적인 구성 설정이 포함되어 있습니다.

core.sleepInterval

core.sleepInterval은 기능 검색 또는 재검색의 연속 통과 간격과 노드 레이블 재지정 간격을 지정합니다. 양수가 아닌 값은 무한 절전 상태를 의미합니다. 재검색되거나 레이블이 다시 지정되지 않습니다.

이 값은 지정된 경우 더 이상 사용되지 않는 --sleep-interval 명령줄 플래그로 재정의됩니다.

사용 예

```
core:
  sleepInterval: 60s 1
```

기본값은 **60s**입니다.

core.sources

core.sources는 활성화된 기능 소스 목록을 지정합니다. 특수한 값 **all**은 모든 기능 소스를 활성화합니다.

이 값은 지정된 경우 더 이상 사용되지 않는 --sources 명령줄 플래그로 재정의됩니다.

기본값: **[all]**

사용 예

```
core:
  sources:
    - system
    - custom
```

core.labelWhiteList

core.labelWhiteList는 레이블 이름을 기반으로 기능 레이블을 필터링하기 위한 정규식을 지정합니다. 일치하지 않는 레이블은 게시되지 않습니다.

정규 표현식은 레이블의 기반 이름 부분인 '/' 뒤에 있는 이름의 부분과만 일치합니다. 레이블 접두사 또는 네임스페이스가 생략됩니다.

이 값은 지정된 경우 더 이상 사용되지 않는 **--label-whitelist** 명령줄 플래그로 재정의됩니다.

기본값: **null**

사용 예

```
core:
  labelWhiteList: '^cpu-cpuid'
```

core.noPublish

core.noPublish를 **true**로 설정하면 **nfd-master**와의 모든 통신이 비활성화됩니다. 이것은 실질적으로는 드라이런 플래그입니다. **nfd-worker**는 정상적으로 기능 감지를 실행하지만 레이블 요청은 **nfd-master**로 전송되지 않습니다.

이 값은 지정된 경우 **--no-publish** 명령줄 플래그로 재정의됩니다.

예제:

사용 예

```
core:
  noPublish: true 1
```

기본값은 **false**입니다.

core.klog

다음 옵션은 대부분 런타임에 동적으로 조정할 수 있는 로거 구성을 지정합니다.

로거 옵션은 명령줄 플래그를 사용하여 지정할 수도 있으며, 이러한 옵션은 해당 구성 파일 옵션보다 우선합니다.

core.klog.addDirHeader

true로 설정하면 **core.klog.addDirHeader**에서 파일 디렉토리를 로그 메시지의 헤더에 추가합니다.

기본값: **false**

런타임 설정 가능: yes

core.klog.alsologtostderr

표준 오류 및 파일에 기록합니다.

기본값: **false**

런타임 설정 가능: yes

core.klog.logBacktraceAt

로깅이 file:N 행에 도달하면 스택 추적을 출력합니다.

기본값: **empty**

런타임 설정 가능: yes

core.klog.logDir

비어 있지 않은 경우 이 디렉터리에 로그 파일을 작성합니다.

기본값: **empty**

런타임 설정 가능: no

core.klog.logFile

비어 있지 않은 경우 이 로그 파일을 사용합니다.

기본값: **empty**

런타임 설정 가능: no

core.klog.logFileMaxSize

core.klog.logFileMaxSize는 로그 파일의 최대 크기를 정의합니다. 단위는 메가바이트입니다. 값이 **0**인 경우 최대 파일 크기는 무제한입니다.

기본값: **1800**

런타임 설정 가능: no

core.klog.logtostderr

파일 대신 표준 오류에 기록합니다.

기본값: **true**

런타임 설정 가능: yes

core.klog.skipHeaders

core.klog.skipHeaders가 **true**로 설정된 경우 로그 메시지에서 헤더 접두사를 사용하지 않습니다.

기본값: **false**

런타임 설정 가능: yes

core.klog.skipLogHeaders

core.klog.skipLogHeaders가 **true**로 설정된 경우 로그 파일을 열 때 헤더를 사용하지 않습니다.

기본값: **false**

런타임 설정 가능: no

core.klog.stderrthreshold

임계값 이상의 로그는 stderr에 있습니다.

기본값: **2**

런타임 설정 가능: yes

core.klog.v

core.klog.v는 로그 수준 세부 정보 표시의 수치입니다.

기본값: **0**

런타임 설정 가능: yes

core.klog.vmodule

core.klog.vmodule은 파일 필터링된 로깅의 쉼표로 구분된 **pattern=N** 설정 목록입니다.

기본값: **empty**

런타임 설정 가능: **yes**

3.4.2. 소스

source 섹션에는 기능 소스 관련 구성 매개변수가 포함되어 있습니다.

sources.cpu.cpuid.attributeBlacklist

이 옵션에 나열된 **cpuid** 기능만을 공개합니다.

이 값은 **sources.cpu.cpuid.attributeWhitelist**로에 의해 재정의됩니다.

기본값: **[BMI1, BMI2, CLMUL, CMOV, CX16, ERMS, F16C, HTT, LZCNT, MMX, MMXEXT, NX, POPCNT, RDRAND, RDSEED, RDTSCP, SGX, SGXLC, SSE, SSE2, SSE3, SSE4.1, SSE4.2, SSSE3]**

사용 예

```
sources:
  cpu:
    cpuid:
      attributeBlacklist: [MMX, MMXEXT]
```

sources.cpu.cpuid.attributeWhitelist

이 옵션에 나열된 **cpuid** 기능만 게시합니다.

sources.cpu.cpuid.attributeWhitelist는 **sources.cpu.cpuid.attributeBlacklist**보다 우선합니다.

기본값: **empty**

사용 예

```
sources:
  cpu:
    cpuid:
      attributeWhitelist: [AVX512BW, AVX512CD, AVX512DQ, AVX512F, AVX512VL]
```

sources.kernel.kconfigFile

sources.kernel.kconfigFile은 커널 구성 파일의 경로입니다. 비어 있는 경우 NFD는 일반적인 표준 위치에서 검색을 실행합니다.

기본값: **empty**

사용 예

```
sources:
  kernel:
    kconfigFile: "/path/to/kconfig"
```

sources.kernel.configOpts

Source.kernel.configOpts는 기능 레이블로 게시하는 커널 구성 옵션을 나타냅니다.

기본값: **[NO_HZ, NO_HZ_IDLE, NO_HZ_FULL, PREEMPT]**

사용 예

```
sources:
  kernel:
    configOpts: [NO_HZ, X86, DMI]
```

sources.pci.deviceClassWhitelist

sources.pci.deviceClassWhitelist 는 레이블을 게시하는 [PCI 장치 클래스 ID](#) 목록입니다. 메인 클래스로만 (예: **03**) 또는 전체 클래스-하위 클래스 조합(예: **0300**)으로 지정할 수 있습니다. 전자는 모든 하위 클래스가 허용됨을 의미합니다. 레이블 형식은 **deviceLabelFields**를 사용하여 추가로 구성할 수 있습니다.

기본값: **["03", "0b40", "12"]**

사용 예

```
sources:
  pci:
    deviceClassWhitelist: ["0200", "03"]
```

sources.pci.deviceLabelFields

sources.pci.deviceLabelFields 는 기능 레이블의 이름을 구성할 때 사용할 PCI ID 필드 집합입니다. 유효한 필드는 **class,vendor,device,subsystem_vendor** 및 **subsystem_device**입니다.

기본값: **[class, vendor]**

사용 예

```
sources:
  pci:
    deviceLabelFields: [class, vendor, device]
```

위의 설정 예제에서 NFD는 **feature.node.kubernetes.io/pci-<class-id>_<vendor-id>_<device-id>.present=true**와 같은 레이블을 게시합니다.

sources.usb.deviceClassWhitelist

sources.usb.deviceClassWhitelist 는 기능 레이블을 게시하는 USB [장치 클래스 ID](#) 목록입니다. 레이블 형식은 **deviceLabelFields**를 사용하여 추가로 구성할 수 있습니다.

기본값: **["0e", "ef", "fe", "ff"]**

사용 예

```
sources:
  usb:
    deviceClassWhitelist: ["ef", "ff"]
```

sources.usb.deviceLabelFields

sources.usb.deviceLabelFields 는 기능 레이블의 이름을 작성할 USB ID 필드의 집합입니다. 유효한 필드는 **class,vendor, device**입니다.

기본값: **[class, vendor, device]**

사용 예

```
sources:
  pci:
    deviceLabelFields: [class, vendor]
```

위의 config 예제에서 NFD는 **feature.node.kubernetes.io/usb-<class-id>_<vendor-id>.present=true**와 같은 레이블을 게시합니다.

sources.custom

sources.custom 은 사용자별 레이블을 생성하기 위해 사용자 정의 기능 소스에서 처리하는 규칙 목록입니다.

기본값: **empty**

사용 예

```
source:
  custom:
    - name: "my.custom.feature"
      matchOn:
        - loadedKMod: ["e1000e"]
        - pcid:
            class: ["0200"]
            vendor: ["8086"]
```

3.5. NFD 토폴로지 업데이트 프로그램 사용

NFD(Node Feature Discovery) Topology Updater는 작업자 노드에서 할당된 리소스를 검사하는 데 사용됩니다. 이 노드는 영역별로 새 Pod에 할당할 수 있는 리소스를 차지하며, 여기서 영역이 NUMA(Non-Uniform Memory Access) 노드일 수 있습니다. NFD Topology Updater는 정보를 nfd-master에 전달하여 클러스터의 모든 작업자 노드에 해당하는 **NodeResourceTopology** CR(사용자 정의 리소스)을 생성합니다. NFD 토폴로지 업데이트 관리자의 하나의 인스턴스는 클러스터의 각 노드에서 실행됩니다.

NFD에서 토폴로지 업데이트 관리자 작업을 활성화하려면 Node Feature Discovery Operator를 사용하는 섹션에 설명된 대로 **NodeFeatureDiscovery** CR에서 **topologyupdater** 변수를 **true** 로 설정합니다.

3.5.1. NodeResourceTopology CR

NFD Topology Updater를 사용하여 실행하는 경우 NFD는 다음과 같은 노드 리소스 하드웨어 토폴로지에 해당하는 사용자 정의 리소스 인스턴스를 생성합니다.

```
apiVersion: topology.node.k8s.io/v1alpha1
kind: NodeResourceTopology
metadata:
  name: node1
topologyPolicies: ["SingleNUMANodeContainerLevel"]
zones:
  - name: node-0
    type: Node
    resources:
      - name: cpu
        capacity: 20
        allocatable: 16
```

```

    available: 10
  - name: vendor/nic1
    capacity: 3
    allocatable: 3
    available: 3
- name: node-1
  type: Node
  resources:
    - name: cpu
      capacity: 30
      allocatable: 30
      available: 15
    - name: vendor/nic2
      capacity: 6
      allocatable: 6
      available: 6
- name: node-2
  type: Node
  resources:
    - name: cpu
      capacity: 30
      allocatable: 30
      available: 15
    - name: vendor/nic1
      capacity: 3
      allocatable: 3
      available: 3

```

3.5.2. NFD Topology Updater 명령줄 플래그

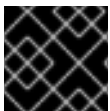
사용 가능한 명령줄 플래그를 보려면 **nfd-topology-updater -help** 명령을 실행합니다. 예를 들어 podman 컨테이너에서 다음 명령을 실행합니다.

```
$ podman run gcr.io/k8s-staging-nfd/node-feature-discovery:master nfd-topology-updater -help
```

-ca-file

-ca-file 플래그는 NFD Topology Updater의 상호 TLS 인증을 제어하는 **-cert-file** 및 **-key-file** 플래그와 함께 세 개의 플래그 중 하나입니다. 이 플래그는 nfd-master의 진위 여부를 확인하는 데 사용되는 TLS 루트 인증서를 지정합니다.

기본값: empty



중요

-ca-file 플래그는 **-cert-file** 및 **-key-file** 플래그와 함께 지정해야 합니다.

예제

```
$ nfd-topology-updater -ca-file=/opt/nfd/ca.crt -cert-file=/opt/nfd/updater.crt -key-file=/opt/nfd/updater.key
```

-cert-file

-cert-file 플래그는 NFD Topology Updater에서 상호 TLS 인증을 제어하는 **-ca-file** 및 **-key-file** 플래그와 함께 세 개의 플래그 중 하나입니다. 이 플래그는 발신 요청을 인증하기 위해 제공되는 TLS 인증서를 지정합니다.

기본값: empty



중요

cert-file 플래그는 **-ca-file** 및 **-key-file** 플래그와 함께 지정해야 합니다.

예제

```
$ nfd-topology-updater -cert-file=/opt/nfd/updater.crt -key-file=/opt/nfd/updater.key -ca-file=/opt/nfd/ca.crt
```

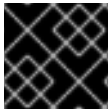
-h, -help

사용법을 출력하고 종료합니다.

-key-file

-key-file 플래그는 NFD Topology Updater의 상호 TLS 인증을 제어하는 **-ca-file** 및 **-cert-file** 플래그와 함께 세 개의 플래그 중 하나입니다. 이 플래그는 발신 요청을 인증하는 데 사용되는 지정된 인증서 파일 또는 **-cert-file**에 해당하는 개인 키를 지정합니다.

기본값: empty



중요

key-file 플래그는 **-ca-file** 및 **-cert-file** 플래그와 함께 지정해야 합니다.

예제

```
$ nfd-topology-updater -key-file=/opt/nfd/updater.key -cert-file=/opt/nfd/updater.crt -ca-file=/opt/nfd/ca.crt
```

-kubelet-config-file

-kubelet-config-file은 Kubelet의 구성 파일의 경로를 지정합니다.

기본값: **/host-var/lib/kubelet/config.yaml**

예제

```
$ nfd-topology-updater -kubelet-config-file=/var/lib/kubelet/config.yaml
```

-no-publish

-no-publish 플래그는 nfd-master와의 모든 통신을 비활성화하여 nfd-topology-updater에 대한 시험 실행 플래그로 설정합니다. NFD Topology Updater는 리소스 하드웨어 토폴로지 탐지를 정상적으로 실행하지만 CR 요청은 nfd-master로 전송되지 않습니다.

기본값: **false**

예제

```
$ nfd-topology-updater -no-publish
```

3.5.2.1. -oneshot

-oneshot 플래그를 사용하면 리소스 하드웨어 토폴로지 탐지를 한 번 통과한 후 NFD Topology Updater가 종료됩니다.

기본값: **false**

예제

```
$ nfd-topology-updater -oneshot -no-publish
```

-podresources-socket

-podresources-socket 플래그는 kubelet이 gRPC 서비스를 내보내는 Unix 소켓의 경로를 지정하여 사용 중인 CPU 및 장치를 검색하고 해당 소켓에 대한 메타데이터를 제공합니다.

기본값: **/host-var/liblib/kubelet/pod-resources/kubelet.sock**

예제

```
$ nfd-topology-updater -podresources-socket=/var/lib/kubelet/pod-resources/kubelet.sock
```

-server

-server 플래그는 연결할 nfd-master 끝점의 주소를 지정합니다.

기본값: **localhost:8080**

예제

```
$ nfd-topology-updater -server=nfd-master.nfd.svc.cluster.local:443
```

-server-name-override

-server-name-override 플래그는 nfd-master TLS 인증서에서 기대할 수 있는 CN(일반 이름)을 지정합니다. 이 플래그는 대부분 개발 및 디버깅용으로 사용됩니다.

기본값: **empty**

예제

```
$ nfd-topology-updater -server-name-override=localhost
```

-sleep-interval

-sleep-interval 플래그는 리소스 하드웨어 토폴로지 재시험 및 사용자 정의 리소스 업데이트 사이의 간격을 지정합니다. 양수가 아닌 값은 무한 절전 간격을 의미하며 재검색이 수행되지 않습니다.

기본값: **60s**

예제

```
$ nfd-topology-updater -sleep-interval=1h
```

-version

버전을 출력하고 종료합니다.

-watch-namespace

-watch-namespace 플래그는 지정된 네임스페이스에서 실행되는 Pod에 대해서만 리소스 하드웨어 토폴로지 검사를 수행하도록 네임스페이스를 지정합니다. 지정된 네임스페이스에서 실행되지 않는 Pod는 리소스 계정 중에 고려되지 않습니다. 이는 테스트 및 디버깅 목적에 특히 유용합니다. * 값은 회계 프로세스 중에 모든 네임스페이스의 모든 Pod를 고려합니다.

기본값: *

예제

```
$ nfd-topology-updater -watch-namespace=rte
```

4장. KERNEL MODULE MANAGEMENT OPERATOR

KMM(커널 모듈 관리) Operator와 OpenShift Container Platform 클러스터에 트리 외부 커널 모듈 및 장치 플러그인을 배포하는 데 사용할 수 있는 방법에 대해 알아봅니다.

4.1. KERNEL MODULE MANAGEMENT OPERATOR 정보

KMM(커널 모듈 관리) Operator는 OpenShift Container Platform 클러스터에서 트리 외부 커널 모듈 및 장치 플러그인을 관리, 빌드, 서명, 배포합니다.

KMM은 트리 외부 커널 모듈 및 관련 장치 플러그인을 설명하는 새 모듈 CRD를 추가합니다. 모듈 리소스를 사용하여 모듈을 로드하는 방법을 구성하고, 커널 버전에 대한 **ModuleLoader** 이미지를 정의하고, 특정 커널 버전에 대한 모듈을 빌드하고 서명하는 지침을 포함할 수 있습니다.

KMM은 모든 커널 모듈에 대해 한 번에 여러 개의 커널 버전을 허용하도록 설계되어 노드를 원활하게 업그레이드하고 애플리케이션 다운타임을 줄일 수 있습니다.

4.2. KERNEL MODULE MANAGEMENT OPERATOR 설치

클러스터 관리자는 OpenShift CLI 또는 웹 콘솔을 사용하여 KMM(커널 모듈 관리) Operator를 설치할 수 있습니다.

KMM Operator는 OpenShift Container Platform 4.12 이상에서 지원됩니다. 버전 4.11에 KMM을 설치하려면 특정 추가 단계가 필요하지 않습니다. 버전 4.10 및 이전 버전에 KMM을 설치하는 방법에 대한 자세한 내용은 "이전 버전의 OpenShift Container Platform에 Kernel Module Management Operator 설치" 섹션을 참조하십시오.

4.2.1. 웹 콘솔을 사용하여 Kernel Module Management Operator 설치

클러스터 관리자는 OpenShift Container Platform 웹 콘솔을 사용하여 KMM(커널 모듈 관리) Operator를 설치할 수 있습니다.

절차

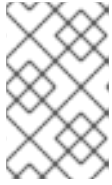
1. OpenShift Container Platform 웹 콘솔에 로그인합니다.
2. Kernel Module Management Operator를 설치합니다.
 - a. OpenShift Container Platform 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.
 - b. 사용 가능한 Operator 목록에서 **Kernel Module Management Operator**를 선택한 다음 **설치**를 클릭합니다.
 - c. **Operator** 설치 페이지에서 **설치 모드**를 클러스터의 특정 네임스페이스로 선택합니다.
 - d. **Installed Namespace** 목록에서 **openshift-kmm** 네임스페이스를 선택합니다.
 - e. **설치**를 클릭합니다.

검증

KMM Operator가 설치되었는지 확인하려면 다음을 수행하십시오.

1. **Operator** → **설치된 Operator** 페이지로 이동합니다.

2. Kernel Module Management Operator 가 openshift-kmm 프로젝트에 InstallSucceeded 상태로 나열되어 있는지 확인합니다.



참고

설치하는 동안 Operator는 실패 상태를 표시할 수 있습니다. 나중에 InstallSucceeded 메시지와 함께 설치에 성공하면 이 실패 메시지를 무시할 수 있습니다.

문제 해결

1. Operator 설치 문제를 해결하려면 다음을 수행합니다.
 - a. Operator → 설치된 Operator 페이지로 이동하고 Operator 서브스크립션 및 설치 계획 탭의 상태에 장애나 오류가 있는지 검사합니다.
 - b. 워크로드 → Pod 페이지로 이동하여 openshift-kmm 프로젝트에서 Pod 로그를 확인합니다.

4.2.2. CLI를 사용하여 Kernel Module Management Operator 설치

클러스터 관리자는 OpenShift CLI를 사용하여 KMM(커널 모듈 관리) Operator를 설치할 수 있습니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터가 있어야 합니다.
- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 권한이 있는 사용자로 OpenShift CLI에 로그인했습니다.

절차

1. openshift-kmm 네임스페이스에 KMM을 설치합니다.
 - a. 다음 Namespace CR을 생성하고 YAML 파일을 저장합니다(예:kmm-namespace.yaml).

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-kmm
```

- b. 다음 OperatorGroup CR을 생성하고 YAML 파일을 저장합니다(예:kmm-op-group.yaml).

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kernel-module-management
  namespace: openshift-kmm
```

- c. 다음 Subscription CR을 생성하고 YAML 파일을 저장합니다(예:kmm-sub.yaml).

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: kernel-module-management
```

```
namespace: openshift-kmm
spec:
  channel: release-1.0
  installPlanApproval: Automatic
  name: kernel-module-management
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: kernel-module-management.v1.0.0
```

d. 다음 명령을 실행하여 서브스크립션 오브젝트를 생성합니다.

```
$ oc create -f kmm-sub.yaml
```

검증

- Operator 배포가 완료되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get -n openshift-kmm deployments.apps kmm-operator-controller-manager
```

출력 예

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
kmm-operator-controller-manager    1/1 1 1 97s
```

Operator를 사용할 수 있습니다.

4.2.3. 이전 버전의 OpenShift Container Platform에 Kernel Module Management Operator 설치

KMM Operator는 OpenShift Container Platform 4.12 이상에서 지원됩니다. 버전 4.10 및 이전 버전의 경우 새 **SecurityContextConstraint** 오브젝트를 생성하여 Operator의 **ServiceAccount**에 바인딩해야 합니다. 클러스터 관리자는 OpenShift CLI를 사용하여 KMM(커널 모듈 관리) Operator를 설치할 수 있습니다.

사전 요구 사항

- 실행 중인 OpenShift Container Platform 클러스터가 있어야 합니다.
- OpenShift CLI(oc)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 OpenShift CLI에 로그인했습니다.

절차

1. **openshift-kmm** 네임스페이스에 KMM을 설치합니다.
 - a. 다음 **Namespace CR**을 생성하고 **YAML** 파일을 저장합니다(예: **kmm-namespace.yaml**).

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-kmm
```


- b. 다음 **SecurityContextConstraint** 오브젝트를 생성하고 YAML 파일을 저장합니다(예:kmm-security-constraint.yaml:).

```

allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups: []
kind: SecurityContextConstraints
metadata:
  name: restricted-v2
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret

```

- c. 다음 명령을 실행하여 **SecurityContextConstraint** 오브젝트를 Operator의 **ServiceAccount**에 바인딩합니다.

```
$ oc apply -f kmm-security-constraint.yaml
```

```
$ oc adm policy add-scc-to-user kmm-security-constraint -z kmm-operator-controller-manager -n openshift-kmm
```

- d. 다음 **OperatorGroup** CR을 생성하고 YAML 파일을 저장합니다(예:kmm-op-group.yaml).

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:

```

```
name: kernel-module-management
namespace: openshift-kmm
```

- e. 다음 Subscription CR을 생성하고 YAML 파일을 저장합니다(예:kmm-sub.yaml).

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: kernel-module-management
  namespace: openshift-kmm
spec:
  channel: release-1.0
  installPlanApproval: Automatic
  name: kernel-module-management
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: kernel-module-management.v1.0.0
```

- f. 다음 명령을 실행하여 서브스크립션 오브젝트를 생성합니다.

```
$ oc create -f kmm-sub.yaml
```

검증

- Operator 배포가 완료되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get -n openshift-kmm deployments.apps kmm-operator-controller-manager
```

출력 예

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
kmm-operator-controller-manager 1/1 1 1 97s
```

Operator를 사용할 수 있습니다.

4.3. 커널 모듈 관리(KMM)에서 서명 사용

Secure Boot가 활성화된 시스템에서 모든 커널 모듈(kmods)은 MOK(Machine Owner's Key) 데이터베이스에 등록된 공개/개인 키 쌍으로 서명해야 합니다. 배포판의 일부로 배포된 드라이버는 이미 배포의 개인 키로 서명해야 하지만 커널 모듈 빌드의 경우 KMM은 커널 매핑의 **sign** 섹션을 사용하여 커널 모듈 서명을 지원합니다.

Secure Boot 사용에 대한 자세한 내용은 [공개 및 개인 키 쌍](#) 생성을 참조하십시오.

사전 요구 사항

- 올바른 (DER) 형식의 공개 개인 키 쌍입니다.
- MOK 데이터베이스에 등록된 공개 키가 있는 하나 이상의 보안 부팅 가능 노드.
- 사전 빌드된 드라이버 컨테이너 이미지 또는 클러스터 내 빌드에 필요한 소스 코드와 **Dockerfile**입니다.

4.4. SECUREBOOT를 위한 키 추가

KMM(KMM)을 사용하여 커널 모듈에 서명하려면 인증서와 개인 키가 필요합니다. 이러한 키 생성 방법에 대한 자세한 내용은 공개 및 개인 키 쌍 생성을 참조하십시오.

공개 키 및 개인 키 쌍을 추출하는 방법에 대한 자세한 내용은 개인 키로 커널 모듈 서명을 참조하십시오. 1~4 단계를 사용하여 파일로 키를 추출합니다.

절차

1. 인증서가 포함된 `sb_cert.cer` 파일과 개인 키가 포함된 `sb_cert.priv` 파일을 만듭니다.

```
$ openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 -batch -config
configuration_file.config -outform DER -out my_signing_key_pub.der -keyout
my_signing_key.priv
```

2. 다음 방법 중 하나를 사용하여 파일을 추가합니다.

- 파일을 시크릿으로 직접 추가합니다.

```
$ oc create secret generic my-signing-key --from-file=key=<my_signing_key.priv>
```

```
$ oc create secret generic my-signing-key-pub --from-file=key=
<my_signing_key_pub.der>
```

- 파일을 base64로 인코딩하여 추가합니다.

```
$ cat sb_cert.priv | base64 -w 0 > my_signing_key2.base64
```

```
$ cat sb_cert.cer | base64 -w 0 > my_signing_key_pub.base64
```

3. 인코딩된 텍스트를 YAML 파일에 추가합니다.

```
apiVersion: v1
kind: Secret
metadata:
  name: my-signing-key-pub
  namespace: default ①
type: Opaque
data:
  cert: <base64_encoded_secureboot_public_key>
```

```
---
apiVersion: v1
kind: Secret
metadata:
  name: my-signing-key
  namespace: default ②
type: Opaque
data:
  key: <base64_encoded_secureboot_private_key>
```

① ① ① ② namespace - default 를 유효한 네임스페이스로 바꿉니다.

4. YAML 파일을 적용합니다.

```
$ oc apply -f <yaml_filename>
```

4.4.1. 키 확인

키를 추가한 후에는 키가 올바르게 설정되었는지 확인해야 합니다.

절차

1. 공개 키 시크릿이 올바르게 설정되었는지 확인합니다.

```
$ oc get secret -o yaml <certificate secret name> | awk '/cert/{print $2; exit}' | base64 -d | openssl x509 -inform der -text
```

Serial Number, Issuer, Subject 등의 인증서가 표시되어야 합니다.

2. 개인 키 보안이 올바르게 설정되었는지 확인합니다.

```
$ oc get secret -o yaml <private key secret name> | awk '/key/{print $2; exit}' | base64 -d
```

-----BEGIN PRIVATE KEY ----- 및 ----- 줄에 묶은 키가 표시되어야 합니다.

4.5. 사전 빌드된 드라이버 컨테이너 서명

하드웨어 벤더에 의해 배포되거나 다른 위치에서 빌드된 이미지와 같이 사전 빌드된 이미지가 있는 경우 이 절차를 사용하십시오.

다음 **YAML** 파일은 공개/개인 키 쌍에 필요한 키 이름(개인 키의 키 키, 공개 키의 인증서)이 있는 시크릿으로 공개/개인 키 쌍을 추가합니다. 그러면 클러스터에서 **unsignedImage** 이미지를 가져와서 열고, **filesToSign**에 나열된 커널 모듈에 서명하고, 다시 추가하고, 결과 이미지를 **containerImage**로 푸시합니다.

그런 다음 서명된 **kmods**를 선택기와 일치하는 모든 노드에 로드하는 **DaemonSet**을 배포해야 합니다. 드라이버 컨테이너는 **MOK** 데이터베이스에 공개 키가 있는 노드와 보안 부팅이 활성화되지 않은 모든 노드에서 서명을 무시해야 합니다. 보안 부팅이 활성화된 상태에서 로드되지 않아야 하지만 **MOK** 데이터베이스에는 해당 키가 없습니다.

사전 요구 사항

- **keySecret** 및 **certSecret** 시크릿이 생성되었습니다.

절차

1.

YAML 파일을 적용합니다.

```

---
apiVersion: kmm.sigs.x-k8s.io/v1beta1
kind: Module
metadata:
  name: example-module
spec:
  moduleLoader:
    serviceAccountName: default
  container:
    modprobe: 1
    moduleName: '<your module name>'
    kernelMappings:
      # the kmods will be deployed on all nodes in the cluster with a kernel that
      # matches the regexp
      - regexp: '^.*\x86_64$'
        # the container to produce containing the signed kmods
        containerImage: <image name e.g. quay.io/myuser/my-driver:<kernelversion>-
signed>
        sign:
          # the image containing the unsigned kmods (we need this because we are not
          # building the kmods within the cluster)
          unsignedImage: <image name e.g. quay.io/myuser/my-driver:<kernelversion> >
          keySecret: # a secret holding the private secureboot key with the key 'key'
            name: <private key secret name>
          certSecret: # a secret holding the public secureboot key with the key 'cert'
            name: <certificate secret name>
          filesToSign: # full path within the unsignedImage container to the kmod(s) to
            sign
            - /opt/lib/modules/4.18.0-348.2.1.el8_5.x86_64/kmm_ci_a.ko
    imageRepoSecret:
      # the name of a secret containing credentials to pull unsignedImage and push
      # containerImage to the registry
      name: repo-pull-secret
    selector:
      kubernetes.io/arch: amd64

```

1

modprobe - 로드할 kmod의 이름입니다.

4.6. MODULELOADER 컨테이너 이미지 빌드 및 서명

소스 코드가 있고 이미지를 먼저 빌드해야 하는 경우 이 절차를 사용하십시오.

다음 **YAML** 파일은 리포지토리의 소스 코드를 사용하여 새 컨테이너 이미지를 빌드합니다. 생성된 이미지는 임시 이름으로 레지스트리에 다시 저장되며 이 임시 이미지는 **sign** 섹션의 매개변수를 사용하여

서명됩니다.

임시 이미지 이름은 최종 이미지 이름을 기반으로 하며 `<containerImage>:<tag>-<namespace>_<module name>_kmm_unsigned` 로 설정됩니다.

예를 들어, 다음 YAML 파일을 사용하여 커널 모듈 관리(KMM)는 서명되지 않은 `kmods`가 포함된 빌드가 포함된 `example.org/repository/minimal-driver:final-default_example-module_kmm_unsigned` 이미지를 빌드하여 레지스트리로 내보냅니다. 그런 다음 서명된 `kmods`가 포함된 `example.org/repository/minimal-driver:final` 이라는 두 번째 이미지를 생성합니다. `DaemonSet` 오브젝트에서 로드하고 `kmods`를 클러스터 노드에 배포하는 두 번째 이미지입니다.

서명된 후에는 레지스트리에서 임시 이미지를 안전하게 삭제할 수 있습니다. 필요한 경우 다시 빌드됩니다.

사전 요구 사항

- `keySecret` 및 `certSecret` 시크릿이 생성되었습니다.

절차

1. YAML 파일을 적용합니다.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-module-dockerfile
  namespace: default ①
data:
  Dockerfile: |
    ARG DTK_AUTO
    ARG KERNEL_VERSION
    FROM ${DTK_AUTO} as builder
    WORKDIR /build/
    RUN git clone -b main --single-branch https://github.com/rh-ecosystem-edge/kernel-
    module-management.git
    WORKDIR kernel-module-management/ci/kmm-kmod/
    RUN make
    FROM registry.access.redhat.com/ubi8/ubi:latest
    ARG KERNEL_VERSION
    RUN yum -y install kmod && yum clean all
    RUN mkdir -p /opt/lib/modules/${KERNEL_VERSION}
    COPY --from=builder /build/kernel-module-management/ci/kmm-kmod/*.ko
    /opt/lib/modules/${KERNEL_VERSION}/
    RUN /usr/sbin/depmod -b /opt
```

```

---
apiVersion: kmm.sigs.x-k8s.io/v1beta1
kind: Module
metadata:
  name: example-module
  namespace: default ❷
spec:
  moduleLoader:
    serviceAccountName: default ❸
  container:
    modprobe:
      moduleName: simple_kmod
    kernelMappings:
      - regexp: '^.*\x86_64$'
        containerImage: < the name of the final driver container to produce>
    build:
      dockerfileConfigMap:
        name: example-module-dockerfile
    sign:
      keySecret:
        name: <private key secret name>
      certSecret:
        name: <certificate secret name>
      filesToSign:
        - /opt/lib/modules/4.18.0-348.2.1.el8_5.x86_64/kmm_ci_a.ko
  imageRepoSecret: ❹
    name: repo-pull-secret
  selector: # top-level selector
    kubernetes.io/arch: amd64

```

❶ ❷

namespace - default 를 유효한 네임스페이스로 바꿉니다.

❸

serviceAccountName - 기본 **serviceAccountName** 에는 권한이 있는 모듈을 실행하는 데 필요한 권한이 없습니다. 서비스 계정 생성에 대한 자세한 내용은 이 섹션의 "해결 리소스"의 "서비스 계정 생성"을 참조하십시오.

❹

imageRepoSecret - **DaemonSet** 오브젝트에서 **imagePullSecrets** 로 사용하고 빌드 및 서명 기능을 가져와서 푸시합니다.

추가 리소스

서비스 계정 생성에 대한 자세한 내용은 서비스 계정 [생성](#)을 참조하십시오.

4.7. 디버깅 및 문제 해결

드라이버 컨테이너의 **kmods**가 서명되지 않았거나 잘못된 키로 서명된 경우 컨테이너는 **PostStartHookError** 또는 **CrashLoopBackOff** 상태를 입력할 수 있습니다. 이 시나리오에 다음 메시지를 표시하는 컨테이너에서 **oc describe** 명령을 실행하여 확인할 수 있습니다.

```
modprobe: ERROR: could not insert '<your_kmod_name>': Required key not available
```