

# **OpenShift Container Platform 4.12**

# 특수 하드웨어 및 드라이버 활성화

OpenShift Container Platform의 하드웨어 활성화 관련 정보

Last Updated: 2024-09-07

OpenShift Container Platform의 하드웨어 활성화 관련 정보

# 법적 공지

Copyright © 2024 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 <sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java <sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

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

MySQL <sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js <sup>®</sup> 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 <sup>®</sup> 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 설치	11
3.2. NODE FEATURE DISCOVERY OPERATOR 사용	13
3.3. NODE FEATURE DISCOVERY OPERATOR 설정	19
3.4. NODEFEATURERULE 사용자 정의 리소스 정보	24
3.5. NODEFEATURERULE 사용자 정의 리소스 사용	24
3.6. NFD 토폴로지 업데이트 프로그램 사용	25
4장. KERNEL MODULE MANAGEMENT OPERATOR	30
4.1. KERNEL MODULE MANAGEMENT OPERATOR 정보	30
4.2. KERNEL MODULE MANAGEMENT OPERATOR 설치	30
4.3. 커널 모듈 배포	34
4.4. MODULELOADER 이미지 사용	38
4.5. 커널 모듈 관리(KMM)에서 서명 사용	41
4.6. SECUREBOOT를 위한 키 추가	41
4.7. 사전 빌드된 드라이버 컨테이너 서명	43
4.8. MODULELOADER 컨테이너 이미지 빌드 및 서명	44
4.9. 디버깅 및 문제 해결	46
4.10. KMM 펌웨어 지원	47
4.11. KMM 문제 해결	49
4.12. KMM HUB 및 SPOKE	53

# 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 설치하 는 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 RHCOS(Red Hat Enterprise Linux CoreOS)와 같은 컨테이너 운영 체제에서 트리 외부 커널 모듈 및 드라이버를 빌드하고 배포하는 데 사용되는 컨테이너 이미지입니다. 커널 모듈과 드라 이버는 운영 체제 커널에서 높은 수준의 권한으로 실행되는 소프트웨어 라이브러리입니다. 커널 기능을 확장하거나 새 장치를 제어하는 데 필요한 하드웨어별 코드를 제공합니다. 예를 들면 필드 프로그래밍 가 능 게이트 어레이(FPGA) 또는 그래픽 처리 장치(GPU)와 같은 하드웨어 장치 및 모두 클라이언트 시스템 에 커널 모듈이 필요한 소프트웨어 정의 스토리지 솔루션이 있습니다. 드라이버 컨테이너는 OpenShift Container Platform 배포에서 이러한 기술을 활성화하는 데 사용되는 소프트웨어 스택의 첫 번째 계층입 니다.

# 2장. 드라이버 툴킷

Driver Toolkit과 OpenShift Container Platform 배포에서 특수 소프트웨어 및 하드웨어 장치를 활성화하 기 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 이미지에는 커널 모듈을 빌드하거나 설치하 는 데 일반적으로 필요한 커널 패키지와 드라이버 컨테이너에 필요한 몇 가지 툴이 포함되어 있습니다. 이 러한 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 빌드하 고 배포하는 데 사용되는 컨테이너 이미지입니다. 커널 모듈과 드라이버는 운영 체제 커널에서 높은 수준 의 권한으로 실행되는 소프트웨어 라이브러리입니다. 커널 기능을 확장하거나 새 장치를 제어하는 데 필 요한 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 스토리 지(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 도구 가 있습니다.

- elfutils-libelf-devel
- kmod
- binutilskabi-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)에서 도 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 또 는 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 배 포할 수 있습니다.

## 2.2. DRIVER TOOLKIT 컨테이너 이미지 가져오기

**driver-toolkit** 이미지는 Red Hat Ecosystem Catalog의 컨테이너 이미지 섹션과 OpenShift Container Platform 릴리스 페이로드에서 사용할 수 있습니다. OpenShift Container Platform의 최신 마이너 릴리스 에 해당하는 이미지에는 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** 이미지 를 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. 페이로드에서 Driver Toolkit 이미지 URL 검색

사전 요구 사항

- Red Hat OpenShift Cluster Manager에서 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-artdev@sha256:0fd84aee79606178b6561ac71f8540f404d518ae5deff45f6d6ac8f02636c7f4

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

\$ podman pull --authfile=path/to/pullsecret.json quay.io/openshift-release-dev/ocp-v4.0-artdev@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 Managed로 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

 YAML은 simple-kmod 드라이버 컨테이너 이미지를 저장하기 위한 ImageStream과 컨테이너 빌 드를 위한 BuildConfig를 정의합니다. 이 YAML을 0000-buildconfig.yaml.template로 저장합니 다.

```
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-artdev@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=drivertoolkit)

\$ sed "s#DRIVER\_TOOLKIT\_IMAGE#\${DRIVER\_TOOLKIT\_IMAGE}#" 0000buildconfig.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 DaemonSet으로 배포합니다.
  - a. 호스트에서 커널 모듈을 로드하려면 드라이버 컨테이너를 권한 있는 보안 컨텍스트로 실행해 야 합니다. 다음 YAML 파일에는 RBAC 규칙과 드라이버 컨테이너 실행을 위한 DaemonSet이 포함되어 있습니다. 이 YAML을 1000-drivercontainer.yaml로 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-kmoddemo/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 제 대로 로드되었는지 확인합니다.

a. pod가 실행 중인지 확인합니다.

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

출력 예

NAMEREADYSTATUSRESTARTSAGEsimple-kmod-driver-build-1-build0/1Completed06msimple-kmod-driver-container-b22fd1/1Running040ssimple-kmod-driver-container-jz9vn1/1Running040ssimple-kmod-driver-container-p45cc1/1Running040s

b. 드라이버 컨테이너 Pod에서 **Ismod** 명령을 실행합니다.

\$ oc exec -it pod/simple-kmod-driver-container-p45cc -- Ismod | grep simple

출력 예

simple\_procfs\_kmod 16384 0 simple\_kmod 16384 0

- 2.4. 추가 리소스
  - 클러스터의 레지스트리 스토리지 구성에 대한 자세한 내용은 OpenShift Container Platform의 이 미지 레지스트리 Operator 를 참조하십시오.

# 3장. NODE FEATURE DISCOVERY OPERATOR

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

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.

## 3.1. NODE FEATURE DISCOVERY OPERATOR 설치

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

## 3.1.1. 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 생성 하고 nfd-namespace.yaml 파일에 YAML을 저장합니다. cluster-monitoring 를 "true" 로 설정합니다.

apiVersion: v1 kind: Namespace metadata: name: openshift-nfd labels: name: openshift-nfd openshift.io/cluster-monitoring: "true"

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 저장합 니다.

```
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.



## 검증

• Operator 배포가 완료되었는지 확인하려면 다음을 실행합니다.

\$ oc get pods

출력 예

NAMEREADYSTATUSRESTARTSAGEnfd-controller-manager-7f86ccfb58-vgr4x2/2Running010m

성공적인 배포에는 Running 상태가 표시됩니다.

3.1.2. 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 Discovery를 선택한 다음 설치를 클릭합니다.
- 3. Operator 설치 페이지에서 클러스터의 특정 네임스페이스를 선택한 다음 설치를 클릭합니다. 네 임스페이스가 생성되므로 생성할 필요가 없습니다.

## 검증

다음과 같이 NFD Operator가 설치되었는지 확인합니다.

- 1. Operator → 설치된 Operator 페이지로 이동합니다.
- 2. Node Feature Discovery가 openshift-nfd 프로젝트에 InstallSucceeded 상태로 나열되어 있는 지 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 상 태에 장애나 오류가 있는지 검사합니다.
- 2. Workloads → Pod 페이지로 이동하여 openshift-nfd 프로젝트에서 Pod 로그를 확인합니다.

## 3.2. NODE FEATURE DISCOVERY OPERATOR 사용

NFD(**Node Feature Discovery**) Operator는 NodeFeatureDiscovery CR(사용자 정의 리소스)을 확인하 여 Node-Feature-Discovery 데몬 세트를 실행하는 데 필요한 모든 리소스를 오케스트레이션합니다. **NodeFeatureDiscovery** CR을 기반으로 Operator는 선택한 네임스페이스에 피연산자(NFD) 구성 요소 를 생성합니다. CR을 편집하여 다른 옵션 중에서 다른 네임스페이스, 이미지, 이미지 가져오기 정책 및 nfd-worker-conf 구성 맵을 사용할 수 있습니다.

클러스터 관리자는 OpenShift CLI(**oc**) 또는 웹 콘솔을 사용하여 **NodeFeatureDiscovery** CR을 생성할 수 있습니다.

## 3.2.1. CLI를 사용하여 NodeFeatureDiscovery CR 생성

클러스터 관리자는 OpenShift CLI(**oc**)를 사용하여 **NodeFeatureDiscovery** CR 인스턴스를 생성할 수 있 습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(**oc**)를 설치합니다.

- cluster-admin 권한이 있는 사용자로 로그인했습니다.
- NFD Operator가 설치되어 있어야 합니다.

## 절차

1. NodeFeatureDiscovery CR을 생성합니다.

## NodeFeatureDiscovery CR의 예

```
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"
```



2. 다음 명령을 실행하여 **NodeFeatureDiscovery** CR을 생성합니다.

\$ oc apply -f <filename>

## 검증

1. 다음 명령을 실행하여 NodeFeatureDiscovery CR이 생성되었는지 확인합니다.

\$ oc get pods

출력 예

NAME	READY	STATUS F	RESTARTS	AGE
nfd-controller-manager-7f86c	cfb58-vgr	4x 2/2 Ru	nning 0	11m
nfd-master-hcn64	1/1	Running 0	60s	
nfd-master-Innxx	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.

## 3.2.2. 연결이 끊긴 환경에서 CLI를 사용하여 NodeFeatureDiscovery CR 생성

클러스터 관리자는 OpenShift CLI(**oc**)를 사용하여 **NodeFeatureDiscovery** CR 인스턴스를 생성할 수 있 습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 권한이 있는 사용자로 로그인했습니다.
- NFD Operator가 설치되어 있어야 합니다.
- 필요한 이미지가 있는 미러 레지스트리에 액세스할 수 있습니다.
- skopeo CLI 툴을 설치했습니다.

## 절차

- 1. 레지스트리 이미지의 다이제스트를 확인합니다.
  - a. 다음 명령을 실행합니다.

\$ skopeo inspect docker://registry.redhat.io/openshift4/ose-node-feature-discovery: <openshift\_version>

명령 예

\$

- \$ skopeo inspect docker://registry.redhat.io/openshift4/ose-node-feature-discovery:v4.12
- b. 출력을 검사하여 이미지 다이제스트를 식별합니다.

```
출력 예
{
...
"Digest":
"sha256:1234567890abcdef1234567890abcdef1234567890abcdef1
...
}
```

2. 다음 명령을 실행하여 **skopeo** CLI 툴을 사용하여 **registry.redhat.io** 의 이미지를 미러 레지스트 리로 복사합니다.

skopeo copy docker://registry.redhat.io/openshift4/ose-node-featurediscovery@<image\_digest> docker://<mirror\_registry>/openshift4/ose-node-featurediscovery@<image\_digest>

명령 예

```
skopeo copy docker://registry.redhat.io/openshift4/ose-node-feature-
discovery@sha256:1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcd
ef docker://<your-mirror-registry>/openshift4/ose-node-feature-
discovery@sha256:1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcd
ef
```

3. NodeFeatureDiscovery CR을 생성합니다.

```
NodeFeatureDiscovery CR의 예
```

- "MMXEXT"

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
 name: nfd-instance
spec:
 operand:
  image: <mirror registry>/openshift4/ose-node-feature-discovery@<image digest>
  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"
```



4. 다음 명령을 실행하여 **NodeFeatureDiscovery** CR을 생성합니다.

\$ oc apply -f <filename>

## 검증

1. 다음 명령을 실행하여 NodeFeatureDiscovery CR의 상태를 확인합니다.

\$ oc get nodefeaturediscovery nfd-instance -o yaml

2. 다음 명령을 실행하여 ImagePullBackOff 오류 없이 포드가 실행 중인지 확인합니다.

\$ oc get pods -n <nfd\_namespace>

## 3.2.3. 웹 콘솔을 사용하여 NodeFeatureDiscovery CR 생성

클러스터 관리자는 OpenShift Container Platform 웹 콘솔을 사용하여 **NodeFeatureDiscovery** CR을 생 성할 수 있습니다.

## 사전 요구 사항

• OpenShift Container Platform 클러스터에 액세스할 수 있습니다.

- cluster-admin 권한이 있는 사용자로 로그인했습니다.
- NFD Operator가 설치되어 있어야 합니다.

## 절차

- 1. Operator → 설치된 Operator 페이지로 이동합니다.
- 2. Node Feature Discovery 섹션에서 제공된 API 에서 Create instance 를 클릭합니다.
- 3. NodefeatureatureDiscovery CR의 값을 편집합니다.
- 4. 생성을 클릭합니다.

## 3.3. NODE FEATURE DISCOVERY OPERATOR 설정

#### 3.3.1. 코어

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

#### core.sleepInterval

**core.sleepInterval**은 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

기본값은 **60s**입니다.

core.sources core.sources는 활성화된 기능 소스 목록을 지정합니다. 특수한 값 all은 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.labelWhiteList**는 레이블 이름을 기반으로 기능 레이블을 필터링하기 위한 정규식을 지정합니다. 일 치하지 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 true로 설정하면 nfd-master와의 모든 통신이 비활성화됩니다. 이것은 실질적으로는 드라이런 플래그입니다. nfd-worker는 정상적으로 기능 감지를 실행하지만 레이블 요청은 nfd-master로 전송되지 않습니다.

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

예제:

사용 예

core: noPublish: true

기본값은 false입니다.

#### 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 우선 합니다.

#### core.klog.addDirHeader

true로 설정하면 core.klog.addDirHeader에서 파일 디렉터리를 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.logFileMaxSize**는 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 true로 설정된 경우 로그 메시지에서 헤더 접두사를 사용하지 않습니다.

기본값: false

런타임 설정 가능: yes

core.klog.skipLogHeaders core.klog.skipLogHeaders가 true로 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.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

## 3.3.2. 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.attributeWhitelist로에 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.attributeBlacklist보다 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 위치 에서 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 클래스로 만 (예: 03) 또는 전체 클래스-하위 클래스 조합(예: 0300)으로 지정할 수 있습니다. 전자는 모든 하위 클래 스가 허용됨을 의미합니다. 레이블 형식은 deviceLabelFields를 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. 유 효한 필드는 class,vendor,device,subsystem\_vendor 및 subsystem\_device입니다.

```
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=true와 같은 레이블을 게시합니다.

#### sources.usb.deviceClassWhitelist

sources.usb.deviceClassWhitelist 는 기능 레이블을 게시하는 USB 장치 클래스 ID 목록입니다. 레이블 형식은 deviceLabelFields를 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 필 드는 class,vendor, device입니다.

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 목록입 니다.

기본값: empty

사용 예

```
source:

custom:

- name: "my.custom.feature"

matchOn:

- loadedKMod: ["e1000e"]

- pcild:

class: ["0200"]

vendor: ["8086"]
```

## **3.4. NODEFEATURERULE** 사용자 정의 리소스 정보

NodeFeatureRule 오브젝트는 노드의 규칙 기반 사용자 지정 레이블을 위해 설계된 NodeFeatureDiscovery 사용자 정의 리소스입니다. 일부 사용 사례에는 장치에 대한 특정 레이블을 생성 하기 위해 하드웨어 벤더의 애플리케이션별 레이블 지정 또는 배포가 포함됩니다.

NodeFeatureRule 오브젝트는 벤더 또는 애플리케이션별 레이블 및 테인트를 생성하는 방법을 제공합니 다. 유연한 규칙 기반 메커니즘을 사용하여 레이블 및 노드 기능을 기반으로 하는 테인트를 생성합니다.

## 3.5. NODEFEATURERULE 사용자 정의 리소스 사용

규칙 집합이 조건과 일치하는 경우 노드에 레이블을 지정하는 NodeFeatureRule 오브젝트를 생성합니다.

절차

1. 다음 텍스트가 포함된 nodefeaturerule.yaml 이라는 사용자 정의 리소스 파일을 생성합니다.

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureRule
metadata:
 name: example-rule
spec:
 rules:
  - name: "example rule"
   labels:
     "example-custom-feature": "true"
   # Label is created if all of the rules below match
   matchFeatures:
     # Match if "veth" kernel module is loaded
    - feature: kernel.loadedmodule
      matchExpressions:
       veth: {op: Exists}
     # Match if any PCI device with vendor 8086 exists in the system
```

 feature: pci.device matchExpressions: vendor: {op: In, value: ["8086"]}

이 사용자 정의 리소스는 **veth** 모듈이 로드되고 공급 업체 코드 **8086** 이 있는 모든 PCI 장치가 클 러스터에 존재할 때 레이블링이 수행되도록 지정합니다.

2. 다음 명령을 실행하여 nodefeaturerule.yaml 파일을 클러스터에 적용합니다.

\$ oc apply -f https://raw.githubusercontent.com/kubernetes-sigs/node-featurediscovery/v0.13.6/examples/nodefeaturerule.yaml

이 예제에서는 **veth** 모듈이 로드되고 벤더 코드 **8086**이 있는 모든 PCI 장치가 있는 노드에 기능 레이블을 적용합니다.



참고 최대 1분 동안 레이블을 다시 지정할 수 있습니다.

## 3.6. NFD 토폴로지 업데이트 프로그램 사용

NFD(Node Feature Discovery) Topology Updater는 작업자 노드에서 할당된 리소스를 검사하는 데몬입 니다. 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 생성합니 다. NFD 토폴로지 업데이트 관리자의 하나의 인스턴스는 클러스터의 각 노드에서 실행됩니다.

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

## 3.6.1. 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

## 3.6.2. 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'flags와 함 께 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 -keyfile=/opt/nfd/updater.key

#### -cert-file

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

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 -cafile=/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 함 께 세 개의 플래그 중 하나입니다. 이 플래그는 발신 요청을 인증하는 데 사용되는 지정된 인증서 파일 또 는 -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 -cafile=/opt/nfd/ca.crt

## -kubelet-config-file -kubelet-config-file 은 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 실 행 플래그로 설정합니다. NFD Topology Updater는 리소스 하드웨어 토폴로지 탐지를 정상적으로 실행하 지만 CR 요청은 nfd-master로 전송되지 않습니다.

## 기본값: false

예제

\$ nfd-topology-updater -no-publish

## 3.6.2.1. -oneshot

-oneshot 플래그를 사용하면 리소스 하드웨어 토폴로지 탐지를 한 번 통과한 후 NFD Topology Updater 가 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 지정합니 다. 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 간격 을 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 토폴 로지 검사를 수행하도록 네임스페이스를 지정합니다. 지정된 네임스페이스에서 실행되지 않는 Pod는 리 소스 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 장 치 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** 리소스 를 사용하여 모듈을 로드하는 방법을 구성하고, 커널 버전에 대한 **ModuleLoader** 이미지를 정의하고, 특 정 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 업 그레이드하고 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 설치하려 면 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" 섹션 을 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 **설** 치를 클릭합니다.
  - c. Operator 설치 페이지에서 설치 모드를 클러스터의 특정 네임스페이스 로 선택합니다.
  - d. Installed Namespace 목록에서 openshift-kmm 네임스페이스를 선택합니다.
  - e. 설치를 클릭합니다.

## 검증

KMM Operator가 설치되었는지 확인하려면 다음을 수행하십시오.

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

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



참고

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

## 문제 해결

- 1. Operator 설치 문제를 해결하려면 다음을 수행합니다.
  - a. Operator →설치된 Operator 페이지로 이동하고 Operator 서브스크립션 및설치 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

NAMEREADY UP-TO-DATEAVAILABLE AGEkmm-operator-controller-manager1/1197s

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 경 우 새 SecurityContextConstraint 오브젝트를 생성하여 Operator의 ServiceAccount 에 바인딩해야 합니 다. 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 SecurityContextConstraint 오브젝트를 생성하고 YAML 파일을 저장합니다(예:kmmsecurity-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-operatorcontroller-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:



각 모듈 리소스에 대해 커널 모듈 관리(KMM)에서 여러DaemonSet 리소스를 생성할 수 있습니다.

- 클러스터에서 실행되는 호환 커널 버전당 하나의 ModuleLoader DaemonSet.
- 구성된 경우 하나의 장치 플러그인 DaemonSet.

모듈 로더 데몬 세트 리소스는 ModuleLoader 이미지를 실행하여 커널 모듈을 로드합니다. 모듈 로더 이미 지는 .ko 파일과 modprobe 및 sleep 바이너리가 모두 포함된 OCI 이미지입니다.

모듈 로더 Pod가 생성되면 Pod는 modprobe 를 실행하여 지정된 모듈을 커널에 삽입합니다. 그런 다음 종 료될 때까지 유휴 상태로 전환됩니다. 이 경우 ExecPreStop 후크는 modprobe -r 을 실행하여 커널 모듈을 언로드합니다.

.spec.devicePlugin 특성이 모듈 리소스에서 구성된 경우 KMM은 클러스터에장치 플러그인 데몬 세트를 생성합니다. 해당 데몬 세트 대상:

• 모듈 리소스의 .spec.selector 와 일치하는 노드입니다.

• 커널 모듈이 로드된 노드(모듈 로더 Pod가 Ready 상태에 있음).

4.3.1. 모듈 사용자 정의 리소스 정의

CRD( 모듈 사용자 정의 리소스 정의)는 모듈 로더 이미지를 통해 클러스터의 모든 노드 또는 선택 노드에 로 드할 수 있는 커널 모듈을 나타냅니다. Module CR(사용자 정의 리소스)은 호환되는 하나 이상의 커널 버전 과 노드 선택기를 지정합니다.

모듈 리소스에 호환되는 버전은.spec.moduleLoader.container.kernelMappings 아래에 나열됩니다. 커 널 매핑은 리터럴 버전과 일치하거나 regexp 를 사용하여 여러 항목을 동시에 일치시킬 수 있습니다.

Module 리소스의 조정 루프는 다음 단계를 실행합니다.

- 1. .spec.selector 와 일치하는 모든 노드를 나열합니다.
- 2. 해당 노드에서 실행 중인 모든 커널 버전 세트를 빌드합니다.
- 3. 각 커널 버전에 대해 다음을 수행합니다.
  - a. .spec.moduleLoader.container.kernelMappings 를 통과하여 적절한 컨테이너 이미지 이름 을 찾습니다. 커널 매핑에 빌드 또는서명이 정의되어 있고 컨테이너 이미지가 아직 없는 경우 필 요에 따라 빌드, 서명 작업 또는 둘 다 실행합니다.
  - b. 이전 단계에서 결정된 컨테이너 이미지를 사용하여 모듈 로더 데몬 세트를 생성합니다.
  - c. .spec.devicePlugin 이 정의된 경우.spec.devicePlugin.container 에 지정된 구성을 사용하여 장치 플러그인 데몬 세트를 생성합니다.
- 4. 다음과 같이 가비지 수집 을 실행합니다.
  - a. 클러스터의 노드에서 실행하지 않는 커널 버전을 대상으로 하는 기존 데몬 세트 리소스입니다.
  - b. 성공적인 빌드 작업.
  - c. 성공적인 서명 작업.

4.3.2. 보안 및 권한



## 중요

커널 모듈을 로드하는 것은 매우 민감한 작업입니다. 커널 모듈이 로드되면 노드에서 모든 종 류의 작업을 수행할 수 있는 모든 권한이 제공됩니다.

4.3.2.1. ServiceAccounts 및 SecurityContextConstraints

KMM(커널 모듈 관리)은 노드에 커널 모듈을 로드하는 권한 있는 워크로드를 생성합니다. 해당 워크로드에 는 권한 있는 SCC(SecurityContextConstraint) 리소스를 사용할 수 있는ServiceAccounts 가 필요합니 다.

해당 워크로드에 대한 권한 부여 모델은 Module 리소스의 네임스페이스와 해당 사양에 따라 다릅니다.

- .spec.moduleLoader.serviceAccountName 또는 .spec.devicePlugin.serviceAccountName 필드가 설정된 경우 항상 사용됩니다.
- 해당 필드가 설정되지 않은 경우 다음을 수행합니다.

- Operator의 네임 스페이스(기본적으로openshift-kmm )에서 모듈 리소스가 생성되는 경우 KMM은 기본 강력한 ServiceAccounts 를 사용하여 데몬 세트를 실행합니다.
- 모듈 리소스가 다른 네임스페이스에서 생성되는 경우 KMM은 네임스페이스의기본
   ServiceAccount 로 데몬 세트를 실행합니다. 권한 있는 SCC를 사용하도록 수동으로 활성화하지 않는 한 모듈 리소스는 권한 있는 워크로드를 실행할 수 없습니다.



## 중요

OpenShift-kmm 는 신뢰할 수 있는 네임스페이스입니다.

RBAC 권한을 설정할 때 openshift-kmm 네임스페이스에서 모듈 리소스를 생성하는 사용자 또는 ServiceAccount 가 있으면 KMM이 클러스터의 모든 노드에서 권한이 있는 워크로드 를 자동으로 실행합니다.

모든 ServiceAccount 가 권한 있는 SCC를 사용하도록 허용하여 모듈 로더 또는 장치 플러그인 Pod를 실행 하려면 다음 명령을 사용합니다.

\$ oc adm policy add-scc-to-user privileged -z "\${serviceAccountName}" [ -n "\${namespace}" ]

## 4.3.2.2. Pod 보안 표준

OpenShift는 사용 중인 보안 컨텍스트에 따라 네임스페이스 Pod 보안 수준을 자동으로 설정하는 동기화 메 커니즘을 실행합니다. 아무 작업도 필요하지 않습니다.

추가 리소스

• Pod 보안 승인 이해 및 관리

## 4.3.3. 모듈 CR의 예

다음은 주석 처리 모듈 예제입니다.

apiVersion: kmm.sigs.x-k8s.io/v1beta1 kind: Module metadata: name: <my_kmod> spec: moduleLoader: container:</my_kmod>
modprobe:
moduleName: <my_kmod> 1</my_kmod>
dirName: /opt 2
firmwarePath: /firmware 3
parameters: 4
- param=1
kernelMappings: 5
- literal: 6.0.15-300.fc37.x86_64
containerImage: some.registry/org/my-kmod:6.0.15-300.fc37.x86_64
- regexp: '^.+\fc37\.x86_64\$' 6
containerImage: "some.other.registry/org/ <my_kmod>:\${KERNEL_FULL_VERSION}" - regexp: '^.+\$' 7</my_kmod>
containerImage: "some.registry/org/ <my_kmod>:\${KERNEL_FULL_VERSION}"</my_kmod>

build: buildArgs: 8 - name: ARG NAME value: <some value> secrets: - name: <some kubernetes secret> 9 baselmageRegistryTLS: 10 insecure: false insecureSkipTLSVerify: false 11 dockerfileConfigMap: 12 name: <my\_kmod\_dockerfile> sign: certSecret: name: <cert\_secret> 13 keySecret: name: <key secret> 14 filesToSign: - /opt/lib/modules/\${KERNEL\_FULL\_VERSION}/<my\_kmod>.ko registryTLS: 15 insecure: false 16 insecureSkipTLSVerify: false serviceAccountName: <sa module loader> 17 devicePlugin: 18 container: image: some.registry/org/device-plugin:latest (19) env: - name: MY\_DEVICE\_PLUGIN\_ENV\_VAR value: SOME VALUE volumeMounts: 20 - mountPath: /some/mountPath name: <device\_plugin\_volume> volumes: **21** - name: <device plugin volume> configMap: name: <some configmap> serviceAccountName: <sa device plugin> 22 imageRepoSecret: 23 name: <secret name> selector: node-role.kubernetes.io/worker: "" 1 1 1 필수 항목입니다. 2 선택사항: 선택 사항: /firmware/\* 를 노드의 /var/lib/firmware/ 로 복사합니다. 4 선택사항: 하나 이상의 커널 항목이 필요합니다. 정규 표현식과 일치하는 커널을 실행하는 각 노드의 KMM은 \${KERNEL FULL VERSION} 을 사용하

6 여 containerImage 에 지정된 이미지를 실행하는 DaemonSet 리소스를 커널 버전으로 대체합니다.

5

- 7 다른 커널의 경우 my-kmod ConfigMap의 Dockerfile을 사용하여 이미지를 빌드합니다.
- 8 선택사항:
- 9 선택 사항: some-kubernetes-secret 의 값은/run/secrets/some-kubernetes-secret 의 빌드 환경에 서 가져올 수 있습니다.
- 10 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 빌드에서 일반 HTTP를 사용하여 Dockerfile FROM 명령에서 이미지를 가져올 수 있습니다.
- 11 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 빌드에서 일반 HTTP를 사용하여 Dockerfile FROM 명령에서 이미지를 가져올 때 모든 TLS 서버 인증서 검증을 건너뜁니다.
- 12 필수 항목입니다.
- 13 필수: 'cert' 키가 있는 공용 secureboot 키를 보유한 시크릿입니다.
- 14 필수: 키가 'key'인 개인 secureboot 키를 보유한 시크릿입니다.
- 15 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 KMM이 일반 HTTP를 사용하여 컨테이 너 이미지가 이미 존재하는지 확인할 수 있습니다.
- 16 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 KMM은 컨테이너 이미지가 이미 존재 하는지 확인할 때 모든 TLS 서버 인증서 검증을 건너뜁니다.
- 17 선택사항:
- 18 선택사항:
- 19 필수: 장치 플러그인 섹션이 있는 경우
- 20 선택사항:
- 21 선택사항:
- 22 선택사항:
- 23 선택 사항: 모듈 로더 및 장치 플러그인 이미지를 가져오는 데 사용됩니다.

## 4.4. MODULELOADER 이미지 사용

KMM(커널 모듈 관리)은 용도에 맞게 빌드된 모듈 로더 이미지와 함께 작동합니다. 다음은 다음 요구 사항을 충족해야 하는 표준 OCI 이미지입니다.

- .Ko 파일은 /opt/lib/modules/\${KERNEL\_VERSION} 에 있어야 합니다.
- modprobe 및 sleep 바이너리는 \$PATH 변수에 정의해야 합니다.

#### 4.4.1. depmod 실행

모듈 로더 이미지에 여러 커널 모듈이 포함되어 있고 모듈 중 하나가 다른 모듈에 의존하는 경우 빌드 프로세 스 끝에 depmod 를 실행하여 종속성을 생성하고 파일을 매핑하는 것이 가장 좋습니다.



참고

kernel-devel 패키지를 다운로드하려면 Red Hat 서브스크립션이 있어야 합니다.

절차

1. 특정 커널 버전에 대해 modules.dep 및 .map 파일을 생성하려면 depmod -b /opt \${KERNEL\_VERSION}을 실행합니다.

4.4.1.1. Dockerfile 예

OpenShift Container Platform에서 이미지를 빌드하는 경우 Driver Tool Kit (DTK) 사용을 고려하십시 오.

자세한 내용은 권한이 부여된 빌드 사용을 참조하십시오.

```
apiVersion: v1
kind: ConfigMap
metadata:
name: kmm-ci-dockerfile
data:
 dockerfile: |
  ARG DTK AUTO
  FROM ${DTK_AUTO} as builder
  ARG KERNEL VERSION
  WORKDIR /usr/src
  RUN ["git", "clone", "https://github.com/rh-ecosystem-edge/kernel-module-
management.git"]
  WORKDIR /usr/src/kernel-module-management/ci/kmm-kmod
  RUN KERNEL SRC DIR=/lib/modules/${KERNEL VERSION}/build make all
  FROM registry.redhat.io/ubi8/ubi-minimal
  ARG KERNEL VERSION
  RUN microdnf install kmod
  COPY --from=builder /usr/src/kernel-module-management/ci/kmm-kmod/kmm ci a.ko
/opt/lib/modules/${KERNEL VERSION}/
  COPY --from=builder /usr/src/kernel-module-management/ci/kmm-kmod/kmm_ci_b.ko
/opt/lib/modules/${KERNEL_VERSION}/
  RUN depmod -b /opt ${KERNEL VERSION}
```

추가 리소스

• 드라이버 툴킷.

4.4.2. 클러스터의 빌드

KMM은 클러스터에 모듈 로더 이미지를 빌드할 수 있습니다. 다음 지침을 따르십시오.

- 커널 매핑의 빌드 섹션을 사용하여 빌드 지침을 제공합니다.
- 컨테이너 이미지의 Dockerfile 을 dockerfile 키 아래의 ConfigMap 리소스에 복사합니다.
- ConfigMap 이모듈 과 동일한 네임스페이스에 있는지 확인합니다.

KMM은 containerImage 필드에 지정된 이미지 이름이 있는지 확인합니다. 이 경우 빌드를 건너뜁니다.

그러지 않으면 KMM에서 Build 리소스를 생성하여 이미지를 빌드합니다. 이미지가 빌드된 후 KMM은모듈 조정을 진행합니다. 다음 예제를 참조하십시오.

# .... - regexp: '^.+\$' containerImage: "some.registry/org/<my\_kmod>:\${KERNEL\_FULL\_VERSION}" build: buildArgs: 1 - name: ARG NAME value: <some\_value> secrets: 2 - name: <some\_kubernetes\_secret> 3 baseImageRegistryTLS: insecure: false 4 insecureSkipTLSVerify: false 5 dockerfileConfigMap: 6 name: <my\_kmod\_dockerfile> registryTLS: insecure: false 7 insecureSkipTLSVerify: false 8 선택사항: 1 2 선택 사항: 3 은 /run/secrets/some-kubernetes-secret 으로 빌드 Pod에 마운트됩니다. 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 빌드가 일반 HTTP를 사용하여 4 Dockerfile FROM 명령에서 이미지를 가져올 수 있습니다. 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 빌드에서 일반 HTTP를 사용하여 5 Dockerfile FROM 명령에서 이미지를 가져올 때 모든 TLS 서버 인증서 검증을 건너뜁니다. 필수 항목입니다. 6 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 KMM이 일반 HTTP를 사용하여 컨테이 7 너 이미지가 이미 존재하는지 확인할 수 있습니다. 8 선택 사항: 이 매개변수를 사용하지 마십시오. true 로 설정하면 KMM은 컨테이너 이미지가 이미 존재 하는지 확인할 때 모든 TLS 서버 인증서 검증을 건너뜁니다.

## 추가 리소스

• 빌드 구성 리소스.

## 4.4.3. Driver Toolkit 사용

Driver Toolkit (DTK)은 빌드 모듈 로더 이미지를 빌드하기 위한 편리한 기본 이미지입니다. 여기에는 현재 클러스터에서 실행 중인 OpenShift 버전의 툴과 라이브러리가 포함되어 있습니다.

절차

다단계 Dockerfile 의 첫 단계로 DTK를 사용합니다.

- 1. 커널 모듈을 빌드합니다.
- 2. .ko 파일을 ubi-minimal 과 같은 작은 최종 사용자 이미지에 복사합니다.
- 3. 클러스터 내 빌드에서 DTK를 활용하려면 DTK\_AUTO 빌드 인수를 사용합니다. 이 값은Build 리소 스를 생성할 때 KMM에 의해 자동으로 설정됩니다. 다음 예제를 참조하십시오.

**ARG DTK AUTO** FROM \${DTK AUTO} as builder **ARG KERNEL VERSION** WORKDIR /usr/src RUN ["git", "clone", "https://github.com/rh-ecosystem-edge/kernel-modulemanagement.git"] WORKDIR /usr/src/kernel-module-management/ci/kmm-kmod RUN KERNEL\_SRC\_DIR=/lib/modules/\${KERNEL VERSION}/build make all FROM registry.redhat.io/ubi8/ubi-minimal **ARG KERNEL VERSION RUN** microdnf install kmod COPY --from=builder /usr/src/kernel-module-management/ci/kmm-kmod/kmm\_ci\_a.ko /opt/lib/modules/\${KERNEL VERSION}/ COPY --from=builder /usr/src/kernel-module-management/ci/kmm-kmod/kmm ci b.ko /opt/lib/modules/\${KERNEL VERSION}/ **RUN** depmod -b /opt \${KERNEL VERSION}

추가 리소스

• 드라이버 툴킷.

## 4.5. 커널 모듈 관리(KMM)에서 서명 사용

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

Secure Boot 사용에 대한 자세한 내용은 공개 및 개인 키 쌍생성을 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.

## 4.6. 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 포함된sb\_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 시크릿 으로 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=cert= <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 2 namespace - default 를 유효한 네임스페이스로 바꿉니다.

4. YAML 파일을 적용합니다.



**4.6.1.** 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.

4.7. 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 시크 릿으로 공개/개인 키 쌍을 추가합니다. 그러면 클러스터에서 unsignedImage 이미지를 가져와서 열고, filesToSign 에 나열된 커널 모듈에 서명하고, 다시 추가하고, 결과 이미지를 containerImage 로 푸시합 니다.

그런 다음 서명된 kmods를 선택기와 일치하는 모든 노드에 로드하는 DaemonSet을 배포해야 합니다. 드라이버 컨테이너는 MOK 데이터베이스에 공개 키가 있는 노드와 보안 부팅이 활성화되지 않은 모든 노 드에서 서명을 무시해야 합니다. 보안 부팅이 활성화된 상태에서 로드되지 않아야 하지만 MOK 데이터베 이스에는 해당 키가 없습니다.

사전 요구 사항

•

1.

keySecret 및 certSecret 시크릿이 생성되었습니다.

절차

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

modprobe - 로드할 kmod의 이름입니다.

4.8. 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 이 미지는 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.

1

예를 들어, 다음 YAML 파일을 사용하여 커널 모듈 관리(KMM)는 서명되지 않은 kmods가 포함된 빌드 가 포함된 example.org/repository/minimal-driver:final-default\_example-module\_kmm\_unsigned 이미지를 빌드하여 레지스트리로 내보냅니다. 그런 다음 서명된 kmods가 포함된 example.org/repository/minimal-driver:final 이라는 두 번째 이미지를 생성합니다. DaemonSet 오브젝 트에서 로드하고 kmods를 클러스터 노드에 배포하는 두 번째 이미지입니다.

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

사전 요구 사항

•

keySecret 및 certSecret 시크릿이 생성되었습니다.

#### 절차

1.

YAML 파일을 적용합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: example-module-dockerfile
 namespace: default
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 를 유효한 네임스페이스로 바꿉니다.

# 3

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

## 4

imageRepoSecret - DaemonSet 오브젝트에서 imagePullSecrets 로 사용하고 빌드 및 서명 기 능을 가져와서 푸시합니다.

추가 리소스

서비스 계정 생성에 대한 자세한 내용은 서비스 계정 생성 을 참조하십시오.

4.9. 디버깅 및 문제 해결

드라이버 컨테이너의 kmods가 서명되지 않았거나 잘못된 키로 서명된 경우 컨테이너는 PostStartHookError 또는 CrashLoopBackOff 상태를 입력할 수 있습니다. 이 시나리오에 다음 메시지 를 표시하는 컨테이너에서 oc describe 명령을 실행하여 확인할 수 있습니다.

modprobe: ERROR: could not insert '<your\_kmod\_name>': Required key not available

4.10. KMM 펌웨어 지원

커널 모듈은 파일 시스템에서 펌웨어 파일을 로드해야 하는 경우가 있습니다. KMM은 ModuleLoader 이미지에서 노드의 파일 시스템으로 펌웨어 파일 복사를 지원합니다.

modprobe 명령을 실행하여 커널 모듈을 삽입하기 전에 .spec.moduleLoader.container.modprobe.firmwarePath 의 콘텐츠가 노드의 /var/lib/firmware 경로에 복사됩니다.

Pod가 종료되면 modprobe - r 명령을 실행하기 전에 모든 파일과 빈 디렉터리가 해당 위치에서 제거됩 니다.

추가 리소스

모듈 로드 이미지 생성.

4.10.1. 노드에서 조회 경로 구성

OpenShift Container Platform 노드에서 펌웨어의 기본 조회 경로 세트에 /var/lib/firmware 경로가 포함되지 않습니다.

#### 절차

1.

Machine Config Operator를 사용하여 /var/lib/firmware 경로가 포함된 MachineConfig CR(사용자 정의 리소스)을 생성합니다.

apiVersion: machineconfiguration.openshift.io/v1 kind: MachineConfig metadata: labels: machineconfiguration.openshift.io/role: worker name: 99-worker-kernel-args-firmware-path spec: kernelArguments: - 'firmware\_class.path=/var/lib/firmware' 0

필요에 따라 라벨을 구성할 수 있습니다. 단일 노드 OpenShift의 경우 control-pane 또는 master 오브젝트를 사용합니다.

2.

MachineConfig CR을 적용하면 노드가 자동으로 재부팅됩니다.

추가 리소스

Machine Config Operator.

4.10.2. 모듈 로더 이미지 빌드

절차

커널 모듈 자체를 빌드하는 것 외에도 빌더 이미지에 바이너리 펌웨어를 포함합니다.

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

# Build the kmod

RUN ["mkdir", "/firmware"] RUN ["curl", "-o", "/firmware/firmware.bin", "https://artifacts.example.com/firmware.bin"]

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

# Copy the kmod, install modprobe, run depmod

COPY --from=builder /firmware /firmware

## 4.10.3. 모듈 리소스 튜닝

절차

٠

Module CR(사용자 정의 리소스)에서 .spec.moduleLoader.container.modprobe.firmwarePath 를 설정합니다.

apiVersion: kmm.sigs.x-k8s.io/v1beta1 kind: Module metadata: name: my-kmod spec: moduleLoader: container: modprobe: moduleName: my-kmod *# Required* firmwarePath: /firmware 1

선택 사항: /firmware/\* 를 노드의 /var/lib/firmware/ 로 복사합니다.

4.11. KMM 문제 해결

KMM 설치 문제를 해결할 때 로그를 모니터링하여 문제가 발생하는 단계를 확인할 수 있습니다. 그런 다음 해당 단계와 관련된 진단 데이터를 검색합니다.

4.11.1. must-gather 툴 사용

oc adm must-gather 명령은 지원 번들을 수집하고 Red Hat 지원에 디버깅 정보를 제공하는 데 선호 되는 방법입니다. 다음 섹션에 설명된 대로 적절한 인수를 사용하여 명령을 실행하여 특정 정보를 수집합 니다.

추가 리소스

must-gather 툴 정보

4.11.1.1. KMM에 대한 데이터 수집

절차

1.

KMM Operator 컨트롤러 관리자의 데이터를 수집합니다.

a.

deve \_GATHER\_IMAGE 변수를 설정합니다.

\$ export MUST\_GATHER\_IMAGE=\$(oc get deployment -n openshift-kmm kmmoperator-controller-manager -ojsonpath='{.spec.template.spec.containers[? (@.name==''manager'')].env[? (@.name==''RELATED\_IMAGES\_MUST\_GATHER'')].value}') 참고



사용자 정의 네임스페이스에 KMM을 설치한 경우 -n <namespace> 스 위치를 사용하여 네임스페이스를 지정합니다.

b.

must-gather 툴을 실행합니다.

\$ oc adm must-gather --image="\${MUST\_GATHER\_IMAGE}" -- /usr/bin/gather

2.

Operator 로그를 표시합니다.

\$ oc logs -fn openshift-kmm deployments/kmm-operator-controller-manager

예 4.1. 출력 예

```
10228 09:36:37.352405
                         1 request.go:682] Waited for 1.001998746s due to client-
side throttling, not priority and fairness, request:
GET:https://172.30.0.1:443/apis/machine.openshift.io/v1beta1?timeout=32s
10228 09:36:40.767060
                         1 listener.go:44] kmm/controller-runtime/metrics
"msg"="Metrics server is starting to listen" "addr"="127.0.0.1:8080"
10228 09:36:40.769483
                         1 main.go:234] kmm/setup "msg"="starting manager"
10228 09:36:40.769907
                         1 internal.go:366] kmm "msg"="Starting server" "addr"=
{"IP":"127.0.0.1","Port":8080,"Zone":""} "kind"="metrics" "path"="/metrics"
                         1 internal.go:366] kmm "msg"="Starting server" "addr"=
10228 09:36:40.770025
{"IP":"::","Port":8081,"Zone":""} "kind"="health probe"
10228 09:36:40.770128
                         1 leaderelection.go:248] attempting to acquire leader
lease openshift-kmm/kmm.sigs.x-k8s.io...
10228 09:36:40.784396
                         1 leaderelection.go:258] successfully acquired lease
openshift-kmm/kmm.sigs.x-k8s.io
10228 09:36:40.784876
                         1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module" "source"="kind source: *v1beta1.Module"
10228 09:36:40.784925
                         1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module" "source"="kind source: *v1.DaemonSet"
10228 09:36:40.784968
                         1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module" "source"="kind source: *v1.Build"
10228 09:36:40.785001
                         1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module" "source"="kind source: *v1.Job"
                         1 controller.go:185] kmm "msg"="Starting EventSource"
10228 09:36:40.785025
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module" "source"="kind source: *v1.Node"
                         1 controller.go:193] kmm "msg"="Starting Controller"
10228 09:36:40.785039
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="Module"
10228 09:36:40.785458
                         1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="PodNodeModule" "controllerGroup"="" "controllerKind"="Pod"
```

"source"="kind source: *v1.Pod" 10228 09:26:40 786947 1 controllor go:1851 kmm "msg"-"Starting EventSource"
"controller"="PreflightValidation" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidation" "source"="kind source:
*v1beta1.PreflightValidation"
I0228 09:36:40.787406 1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="PreflightValidation" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidation" "source"="kind source: *v1.Build"
I0228 09:36:40.787474 1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="PreflightValidation" "controllerGroup"="kmm.sigs.x-k8s.io"
"ControllerKind"="Prefilght Validation" "Source"="Kind source: ^V1.Job"
10228 09:36:40.787488 I controller.go:185] Kmm msg = Starting EventSource
"controller = Prenigntvalidation ControllerGroup = knim.sigs.x-kos.io
$10228 \ 1028 \ 10228$
"controller"="NodeKernel" "controllerGroup"="" "controllerKind"="Node"
"source"="kind source: *v1.Node"
I0228 09:36:40.787634 1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="NodeKernel" "controllerGroup"="" "controllerKind"="Node"
I0228 09:36:40.787680 1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="PreflightValidation" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidation"
I0228 09:36:40.785607 1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="imagestream" "controllerGroup"="image.openshift.io"
"controllerKind"="ImageStream" "source"="kind source: *v1.ImageStream"
10228 09:36:40.787822 1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="preflightvalidationocp" "controllerGroup"="kmm.sigs.x-k8s.io"
controllerKind = PreflightValidationOCP Source = Kind Source:
10228 00:36:40 787853 1 controller do:1031 kmm "med"-"Starting Controller"
"controller"-"imagestream" "controllerGroup"-"image openshift io"
"controllerKind"="ImageStream"
10228 09:36:40.787879 1 controller.go:1851 kmm "msg"="Starting EventSource"
"controller"="preflightvalidationocp" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidationOCP" "source"="kind source:
*v1beta1.PreflightValidation"
I0228 09:36:40.787905 1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="preflightvalidationocp" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidationOCP"
1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="PodNodeModule" "controllerGroup"="" "controllerKind"="Pod"

## 4.11.1.2. KMM-Hub에 대한 데이터 수집

```
절차
```

1.

KMM Operator hub 컨트롤러 관리자의 데이터를 수집합니다.

a.

deve \_GATHER\_IMAGE 변수를 설정합니다.

참고

\$ export MUST\_GATHER\_IMAGE=\$(oc get deployment -n openshift-kmm-hub kmm-operator-hub-controller-manager ojsonpath='{.spec.template.spec.containers[?(@.name=="manager")].env[? (@.name=="RELATED\_IMAGES\_MUST\_GATHER")].value}')



사용자 정의 네임스페이스에 KMM을 설치한 경우 -n <namespace> 스 위치를 사용하여 네임스페이스를 지정합니다.

```
b.
```

must-gather 툴을 실행합니다.

\$ oc adm must-gather --image="\${MUST\_GATHER\_IMAGE}" -- /usr/bin/gather -u

2.

Operator 로그를 표시합니다.

\$ oc logs -fn openshift-kmm-hub deployments/kmm-operator-hub-controller-manager

```
예 4.2. 출력 예
```

```
10417 11:34:08.807472
                         1 request.go:682] Waited for 1.023403273s due to client-
side throttling, not priority and fairness, request:
GET:https://172.30.0.1:443/apis/tuned.openshift.io/v1?timeout=32s
10417 11:34:12.373413
                         1 listener.go:44] kmm-hub/controller-runtime/metrics
"msg"="Metrics server is starting to listen" "addr"="127.0.0.1:8080"
10417 11:34:12.376253
                         1 main.go:150] kmm-hub/setup "msg"="Adding
controller" "name"="ManagedClusterModule"
10417 11:34:12.376621
                         1 main.go:186] kmm-hub/setup "msg"="starting
manager"
10417 11:34:12.377690
                         1 leaderelection.go:248] attempting to acquire leader
lease openshift-kmm-hub/kmm-hub.sigs.x-k8s.io...
10417 11:34:12.378078
                        1 internal.go:366] kmm-hub "msg"="Starting server"
"addr"={"IP":"127.0.0.1","Port":8080,"Zone":""} "kind"="metrics" "path"="/metrics"
                        1 internal.go:366] kmm-hub "msg"="Starting server"
10417 11:34:12.378222
"addr"={"IP":"::","Port":8081,"Zone":""} "kind"="health probe"
                        1 leaderelection.go:258] successfully acquired lease
10417 11:34:12.395703
openshift-kmm-hub/kmm-hub.sigs.x-k8s.io
                        1 controller.go:185] kmm-hub "msg"="Starting
10417 11:34:12.396334
EventSource" "controller"="ManagedClusterModule"
"controllerGroup"="hub.kmm.sigs.x-k8s.io"
"controllerKind"="ManagedClusterModule" "source"="kind source:
*v1beta1.ManagedClusterModule"
10417 11:34:12.396403
                        1 controller.go:185] kmm-hub "msg"="Starting
EventSource" "controller"="ManagedClusterModule"
"controllerGroup"="hub.kmm.sigs.x-k8s.io"
"controllerKind"="ManagedClusterModule" "source"="kind source:
*v1.ManifestWork"
```

10417 11:34:12.396430 1 controller.go:185] kmm-hub "msg"="Starting EventSource" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" "source"="kind source: \*v1.Build" 10417 11:34:12.396469 1 controller.go:185] kmm-hub "msg"="Starting EventSource" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" "source"="kind source: \*v1.Job" 10417 11:34:12.396522 1 controller.go:185] kmm-hub "msg"="Starting EventSource" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" "source"="kind source: \*v1.ManagedCluster" 10417 11:34:12.396543 1 controller.go:193] kmm-hub "msg"="Starting Controller" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" 10417 11:34:12.397175 1 controller.go:185] kmm-hub "msg"="Starting EventSource" "controller"="imagestream" "controllerGroup"="image.openshift.io" "controllerKind"="ImageStream" "source"="kind source: \*v1.ImageStream" 10417 11:34:12.397221 1 controller.go:193] kmm-hub "msg"="Starting Controller" "controller"="imagestream" "controllerGroup"="image.openshift.io" "controllerKind"="ImageStream" 10417 11:34:12.498335 1 filter.go:196] kmm-hub "msg"="Listing all ManagedClusterModules" "managedcluster"="local-cluster" 10417 11:34:12.498570 1 filter.go:205] kmm-hub "msg"="Listed ManagedClusterModules" "count"=0 "managedcluster"="local-cluster" 10417 11:34:12.498629 1 filter.go:238] kmm-hub "msg"="Adding reconciliation requests" "count"=0 "managedcluster"="local-cluster" 10417 11:34:12.498687 1 filter.go:196] kmm-hub "msg"="Listing all ManagedClusterModules" "managedcluster"="sno1-0" 1 filter.go:205] kmm-hub "msg"="Listed 10417 11:34:12.498750 ManagedClusterModules" "count"=0 "managedcluster"="sno1-0" 1 filter.go:238] kmm-hub "msg"="Adding reconciliation 10417 11:34:12.498801 requests" "count"=0 "managedcluster"="sno1-0" 10417 11:34:12.501947 1 controller.go:227] kmm-hub "msg"="Starting workers" "controller"="imagestream" "controllerGroup"="image.openshift.io" "controllerKind"="ImageStream" "worker count"=1 10417 11:34:12.501948 1 controller.go:227] kmm-hub "msg"="Starting workers" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" "worker count"=1 10417 11:34:12.502285 1 imagestream reconciler.go:50] kmm-hub "msg"="registered imagestream info mapping" "ImageStream"={"name":"drivertoolkit","namespace":"openshift"} "controller"="imagestream" "controllerGroup"="image.openshift.io" "controllerKind"="ImageStream" "dtkImage"="quay.io/openshift-release-dev/ocp-v4.0-artdev@sha256:df42b4785a7a662b30da53bdb0d206120cf4d24b45674227b16051ba4b7 c3934" "name"="driver-toolkit" "namespace"="openshift" "oslmageVersion"="412.86.202302211547-0" "reconcileID"="e709ff0a-5664-4007-8270-49b5dff8bae9"

4.12. KMM HUB 및 SPOKE

허브 및 스포크 시나리오에서 많은 스포크 클러스터는 중앙의 강력한 허브 클러스터에 연결됩니다. KMM(커널 모듈 관리)은 hub 및 spoke 환경에서 작동하는 RHACM(Red Hat Advanced Cluster Management)에 따라 다릅니다.

KMM은 KMM 기능을 분리하여 Hub 및 spoke 환경과 호환됩니다. ManagedClusterModule CRD(Custom Resource Definition)는 기존 모듈 CRD를 래핑하여 스케이크 클러스터를 선택하도록 확 장됩니다. 또한 Hub 클러스터에서 이미지를 빌드하고 서명하는 새로운 독립 실행형 컨트롤러인 KMM-Hub도 제공됩니다.

hub 및 spoke 설정에서 spokes는 hub 클러스터에서 중앙에서 관리하는 리소스 제한 클러스터에 중 점을 두고 있습니다. spokes는 리소스 집약적인 기능이 비활성화된 KMM의 단일 클러스터 버전을 실행 합니다. KMM을 이 환경에 맞게 조정하려면 허브에서 비용이 많이 드는 작업을 처리하는 동안 스포크에서 실행되는 워크로드를 최소화해야 합니다.

커널 모듈 이미지를 빌드하고 .ko 파일에 서명하면 허브에서 실행해야 합니다. 모듈 로뎌 및 장치 플러 그인 DaemonSet의 예약은 spokes 에서만 발생할 수 있습니다.

추가 리소스

• Red Hat Advanced Cluster Management(RHACM)

#### 4.12.1. KMM-Hub

KMM 프로젝트는 허브 클러스터 전용 KMM-Hub 버전을 제공합니다. KMM-Hub는 spokes에서 실행 되는 모든 커널 버전을 모니터링하고 커널 모듈을 수신해야 하는 클러스터의 노드를 결정합니다.

KMM-Hub는 이미지 빌드 및 kmod 서명과 같은 모든 컴퓨팅 집약적인 작업을 실행하고 trimmeddown Module 이 RHACM을 통해 spokes로 전송할 수 있도록 준비합니다.

참고

KMM-Hub는 허브 클러스터에서 커널 모듈을 로드하는 데 사용할 수 없습니다. KMM의 일반 버전을 설치하여 커널 모듈을 로드합니다.

추가 리소스

KMM 설치

## 4.12.2. KMM-Hub 설치

다음 방법 중 하나를 사용하여 KMM-Hub를 설치할 수 있습니다.

- OLM(Operator Lifecycle Manager) 사용
  - KMM 리소스 생성

추가 리소스

- •
- KMM Operator 번들

4.12.2.1. Operator Lifecycle Manager를 사용하여 KMM-Hub 설치

OpenShift 콘솔의 Operators 섹션을 사용하여 KMM-Hub를 설치합니다.

#### 4.12.2.2. KMM 리소스를 생성하여 KMM-Hub 설치

절차

KMM-Hub를 프로그래밍 방식으로 설치하려면 다음 리소스를 사용하여 네임스페이 스,OperatorGroup 및 Subscription 리소스를 생성할 수 있습니다.

apiVersion: v1 kind: Namespace metadata: name: openshift-kmm-hub apiVersion: operators.coreos.com/v1 kind: OperatorGroup metadata: name: kernel-module-management-hub namespace: openshift-kmm-hub apiVersion: operators.coreos.com/v1alpha1 kind: Subscription metadata: name: kernel-module-management-hub namespace: openshift-kmm-hub spec: channel: stable installPlanApproval: Automatic

name: kernel-module-management-hub source: redhat-operators sourceNamespace: openshift-marketplace

## 4.12.3. ManagedClusterModule CRD 사용

ManagedClusterModule CRD(Custom Resource Definition)를 사용하여 spoke 클러스터에서 커널 모듈 배포를 구성합니다. 이 CRD는 클러스터 범위로, 모듈 사양을 래핑하고 다음과 같은 추가 필드를 추 가합니다.



wants-my-mcm: 'true'

1

moduleSpec: 모듈 리소스와 유사하게 moduleLoader 및 devicePlugin 섹션을 포함합니다.

2

ManagedCluster 내에서 노드를 선택합니다.

3

모듈을 생성해야 하는 네임스페이스 를 지정합니다.

4

ManagedCluster 오브젝트를 선택합니다.

.spec.moduleSpec 에 빌드 또는 서명 지침이 있는 경우 해당 Pod는 Operator의 네임스페이스의 허 브 클러스터에서 실행됩니다.

.spec.selector가 하나 이상의 ManagedCluster 리소스와 일치하는 경우 KMM-Hub는 해당 네임스페

이스에 ManifestWork 리소스를 생성합니다. ManifestWork 에는 커널 매핑이 유지되지만 모든 build 및 sign 하위 섹션이 제거된 트리다운 모듈 리소스가 포함되어 있습니다. 태그로 끝나는 이미지 이름이 포함 된 containerImage 필드는 동일한 다이제스트로 교체됩니다.

#### 4.12.4. spoke에서 KMM 실행

spoke에 KMM을 설치한 후에는 추가 작업이 필요하지 않습니다. 허브에서 ManagedClusterModule 오브젝트를 생성하여 spoke 클러스터에 커널 모듈을 배포합니다.

절차

RHACM 정책 오브젝트를 통해 spokes 클러스터에 KMM을 설치할 수 있습니다. Operator 허브에서 KMM을 설치하고 경량 스포크 모드로 실행하는 것 외에도 RHACM 에이전트에서 모듈 리소스를 관리하는 데 필요한 추가 RBAC를 구성합니다.

다음 RHACM 정책을 사용하여 대화 형 클러스티에 KMM을 설치합니다.

apiVersion: policy.open-cluster-management.io/v1 kind: Policy metadata: name: install-kmm spec: remediationAction: enforce disabled: false policy-templates: - objectDefinition: apiVersion: policy.open-cluster-management.io/v1 kind: ConfigurationPolicy metadata: name: install-kmm spec: severity: high object-templates: - complianceType: mustonlyhave objectDefinition: apiVersion: v1 kind: Namespace metadata: name: openshift-kmm - complianceType: mustonlyhave objectDefinition: apiVersion: operators.coreos.com/v1 kind: OperatorGroup metadata: name: kmm namespace: openshift-kmm spec: upgradeStrategy: Default

- complianceType: mustonlyhave objectDefinition: apiVersion: operators.coreos.com/v1alpha1 kind: Subscription metadata: name: kernel-module-management namespace: openshift-kmm spec: channel: stable config: env: - name: KMM\_MANAGED value: "1" installPlanApproval: Automatic name: kernel-module-management source: redhat-operators sourceNamespace: openshift-marketplace - complianceType: mustonlyhave objectDefinition: apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: kmm-module-manager rules: - apiGroups: [kmm.sigs.x-k8s.io] resources: [modules] verbs: [create, delete, get, list, patch, update, watch] - complianceType: mustonlyhave objectDefinition: apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: klusterlet-kmm subjects: - kind: ServiceAccount name: klusterlet-work-sa namespace: open-cluster-management-agent roleRef: kind: ClusterRole name: kmm-module-manager apiGroup: rbac.authorization.k8s.io apiVersion: apps.open-cluster-management.io/v1 kind: PlacementRule metadata: name: all-managed-clusters spec: clusterSelector: 1 matchExpressions: [] apiVersion: policy.open-cluster-management.io/v1 kind: PlacementBinding metadata: name: install-kmm placementRef: apiGroup: apps.open-cluster-management.io

kind: PlacementRule name: all-managed-clusters subjects: - apiGroup: policy.open-cluster-management.io kind: Policy name: install-kmm

1

spec.clusterSelector 필드는 선택한 클러스터만 대상으로 지정하도록 사용자 지정할 수 있습니다.