OpenShift Container Platform 4.12

Matériel spécialisé et activation des pilotes

En savoir plus sur l'activation du matériel sur OpenShift Container Platform

Last Updated: 2023-05-04

OpenShift Container Platform 4.12 Matériel spécialisé et activation des pilotes

En savoir plus sur l'activation du matériel sur OpenShift Container Platform

Notice légale

Résumé

Ce document fournit une vue d'ensemble de l'activation du matériel dans OpenShift Container Platform.

Table des matières

CHAPITRE 1. A PROPOS DU MATÉRIEL SPÉCIALISÉ ET DE L'ACTIVATION DES PILOTES

Le Driver Toolkit (DTK) est une image de conteneur dans la charge utile d'OpenShift Container Platform qui est destinée à être utilisée comme une image de base sur laquelle construire des conteneurs de pilotes. L'image Driver Toolkit contient les paquets du noyau généralement requis comme dépendances pour construire ou installer des modules du noyau, ainsi que quelques outils nécessaires dans les conteneurs de pilotes. La version de ces paquets correspondra à la version du noyau fonctionnant sur les nœuds RHCOS dans la version correspondante d'OpenShift Container Platform.

Les conteneurs de pilotes sont des images de conteneurs utilisées pour construire et déployer des modules de noyau et des pilotes hors arbre sur des systèmes d'exploitation de conteneurs tels que :op-system-first :. Les modules du noyau et les pilotes sont des bibliothèques logicielles fonctionnant avec un niveau de privilège élevé dans le noyau du système d'exploitation. Ils étendent les fonctionnalités du noyau ou fournissent le code spécifique au matériel nécessaire pour contrôler de nouveaux dispositifs. Parmi les exemples, on peut citer les dispositifs matériels tels que les réseaux de portes programmables (FPGA) ou les unités de traitement graphique (GPU), ainsi que les solutions de stockage définies par logiciel, qui nécessitent tous des modules de noyau sur les machines clientes. Les conteneurs de pilotes sont la première couche de la pile logicielle utilisée pour activer ces technologies sur les déploiements d'OpenShift Container Platform.

CHAPITRE 2. DRIVER TOOLKIT

Découvrez le Driver Toolkit et comment vous pouvez l'utiliser comme image de base pour les conteneurs de pilotes afin d'activer des dispositifs logiciels et matériels spéciaux sur les déploiements d'OpenShift Container Platform.

2.1. À PROPOS DU DRIVER TOOLKIT

Contexte

Le Driver Toolkit est une image de conteneur dans la charge utile d'OpenShift Container Platform utilisée comme image de base sur laquelle vous pouvez construire des conteneurs de pilotes. L'image Driver Toolkit comprend les paquets du noyau généralement requis comme dépendances pour construire ou installer des modules du noyau, ainsi que quelques outils nécessaires dans les conteneurs de pilotes. La version de ces paquets correspondra à la version du noyau fonctionnant sur les nœuds Red Hat Enterprise Linux CoreOS (RHCOS) dans la version correspondante d'OpenShift Container Platform.

Les conteneurs de pilotes sont des images de conteneurs utilisées pour construire et déployer des modules de noyau et des pilotes hors arbre sur des systèmes d'exploitation de conteneurs tels que RHCOS. Les modules du noyau et les pilotes sont des bibliothèques logicielles fonctionnant avec un niveau de privilège élevé dans le noyau du système d'exploitation. Ils étendent les fonctionnalités du noyau ou fournissent le code spécifique au matériel nécessaire pour contrôler de nouveaux dispositifs. Les exemples incluent les dispositifs matériels tels que les Field Programmable Gate Arrays (FPGA) ou les GPU, et les solutions de stockage définies par logiciel (SDS), telles que les systèmes de fichiers parallèles Lustre, qui nécessitent des modules de noyau sur les machines clientes. Les conteneurs de pilotes constituent la première couche de la pile logicielle utilisée pour activer ces technologies sur Kubernetes.

La liste des paquets du noyau dans le Driver Toolkit comprend les éléments suivants et leurs dépendances :

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

En outre, le Driver Toolkit comprend également les paquets de noyau en temps réel correspondants :

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

Le Driver Toolkit contient également plusieurs outils qui sont généralement nécessaires pour construire et installer les modules du noyau, notamment :

- **elfutils-libelf-devel**

- **kmod**
- **binutilskabi-dw**
- **kernel-abi-whitelists**
- pour les dépendances ci-dessus

Objectif

Avant l'existence du Driver Toolkit, les utilisateurs installaient les paquets du noyau dans un pod ou construisaient la configuration sur OpenShift Container Platform en utilisant des constructions autorisées ou en installant à partir des RPM du noyau dans les hôtes **machine-os-content**. Le Driver Toolkit simplifie le processus en supprimant l'étape d'habilitation, et évite l'opération privilégiée d'accès à la machine-os-content dans un pod. Le Driver Toolkit peut également être utilisé par les partenaires qui ont accès à des versions pré-vendues d'OpenShift Container Platform pour préconstruire des conteneurs de pilotes pour leurs périphériques matériels pour les futures versions d'OpenShift Container Platform.

Le Driver Toolkit est également utilisé par le Kernel Module Management (KMM), qui est actuellement disponible en tant qu'opérateur communautaire sur OperatorHub. KMM prend en charge les pilotes de noyau hors arbre et tiers, ainsi que le logiciel de support du système d'exploitation sous-jacent. Les utilisateurs peuvent créer des modules pour KMM afin de construire et de déployer un conteneur de pilote, ainsi qu'un logiciel de support tel qu'un plugin de périphérique ou des métriques. Les modules peuvent inclure une configuration de construction pour construire un conteneur de pilote basé sur le Driver Toolkit, ou KMM peut déployer un conteneur de pilote préconstruit.

2.2. EXTRACTION DE L'IMAGE DU CONTENEUR DRIVER TOOLKIT

L'image **driver-toolkit** est disponible dans la section Container images du Red Hat Ecosystem Catalog et dans la release payload d'OpenShift Container Platform. L'image correspondant à la version mineure la plus récente d'OpenShift Container Platform sera étiquetée avec le numéro de version dans le catalogue. L'URL de l'image pour une version spécifique peut être trouvée en utilisant la commande CLI **oc adm**.

2.2.1. Extraction de l'image du conteneur Driver Toolkit à partir de registry.redhat.io

Les instructions pour extraire l'image **driver-toolkit** de **registry.redhat.io** avec **podman** ou dans OpenShift Container Platform peuvent être trouvées sur le Red Hat Ecosystem Catalog . L'image du driver-toolkit pour la dernière version mineure est étiquetée avec la version de la version mineure sur **registry.redhat.io**, par exemple : **registry.redhat.io/openshift4/driver-toolkit-rhel8:v4.12**.

2.2.2. Recherche de l'URL de l'image du Driver Toolkit dans la charge utile

Conditions préalables

- Vous avez obtenu le secret d'extraction d' image à partir du gestionnaire de cluster Red Hat OpenShift.
- You installed the OpenShift CLI (**oc**).

Procédure

1. Utilisez la commande **oc adm** pour extraire l'URL de l'image du site **driver-toolkit** correspondant à une certaine version :

- Pour une image x86, entrez la commande suivante :

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

- Pour une image ARM, entrez la commande suivante :

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

Exemple de sortie

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

2. Obtenez cette image en utilisant un pull secret valide, tel que le pull secret requis pour installer OpenShift Container Platform :

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

2.3. UTILISATION DU DRIVER TOOLKIT

Par exemple, le Driver Toolkit peut être utilisé comme image de base pour construire un module de noyau très simple appelé **simple-kmod**.

NOTE

Le Driver Toolkit inclut les dépendances nécessaires, **openssl**, **mokutil**, et **keyutils**, pour signer un module de noyau. Cependant, dans cet exemple, le module du noyau **simple-kmod** n'est pas signé et ne peut donc pas être chargé sur des systèmes où **Secure Boot** est activé.

2.3.1. Construire et exécuter le conteneur de pilote simple-kmod sur un cluster

Conditions préalables

- Vous disposez d'un cluster OpenShift Container Platform en cours d'exécution.
- Vous avez défini l'état de l'opérateur du registre d'images sur **Managed** pour votre cluster.
- You installed the OpenShift CLI (**oc**).
- Vous êtes connecté à OpenShift CLI en tant qu'utilisateur avec des privilèges **cluster-admin**.

Procédure

Créez un espace de noms. Par exemple :

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

1. Le YAML définit un **ImageStream** pour stocker l'image du conteneur du pilote **simple-kmod** et un **BuildConfig** pour construire le conteneur. Enregistrez ce YAML sous **0000-buildconfig.yaml.template**.

```
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. Remplacez "DRIVER_TOOLKIT_IMAGE" par l'image du driver toolkit correspondant à la version d'OpenShift Container Platform que vous utilisez, à l'aide des commandes suivantes.

```
$ 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. Créer le flux d'images et la configuration de construction avec

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

4. Une fois le pod de construction terminé avec succès, déployez l'image du conteneur de pilote sous la forme d'un fichier **DaemonSet**.

   a. Le conteneur de pilote doit être exécuté avec le contexte de sécurité privilégié afin de charger les modules du noyau sur l'hôte. Le fichier YAML suivant contient les règles RBAC et le site **DaemonSet** pour l'exécution du conteneur de pilote. Enregistrez ce fichier YAML sous **1000-drivercontainer.yaml**.

```
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. Créer les règles RBAC et le jeu de démons :

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

5. Une fois que les pods fonctionnent sur les nœuds de travail, vérifiez que le module du noyau **simple_kmod** est chargé avec succès sur les machines hôtes avec **lsmod**.

a. Vérifiez que les modules sont en cours d'exécution :

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

Exemple de sortie

```
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. Exécutez la commande **lsmod** dans le conteneur de pilotes :

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

Exemple de sortie

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

2.4. RESSOURCES SUPPLÉMENTAIRES

- Pour plus d'informations sur la configuration du stockage du registre pour votre cluster, voir Image Registry Operator in OpenShift Container Platform .

CHAPITRE 3. NODE FEATURE DISCOVERY OPERATOR

Découvrez l'opérateur Node Feature Discovery (NFD) et comment vous pouvez l'utiliser pour exposer des informations au niveau du nœud en orchestrant Node Feature Discovery, un module complémentaire de Kubernetes pour détecter les caractéristiques matérielles et la configuration du système.

3.1. À PROPOS DE L'OPÉRATEUR DE RECHERCHE DE CARACTÉRISTIQUES DE NŒUDS

L'opérateur de découverte des caractéristiques des nœuds (NFD) gère la détection des caractéristiques et de la configuration du matériel dans un cluster OpenShift Container Platform en étiquetant les nœuds avec des informations spécifiques au matériel. NFD étiquette l'hôte avec des attributs spécifiques au nœud, tels que les cartes PCI, le noyau, la version du système d'exploitation, etc.

L'opérateur NFD peut être trouvé sur le Hub de l'opérateur en recherchant "Node Feature Discovery".

3.2. INSTALLATION DE L'OPÉRATEUR DE DÉCOUVERTE DE FONCTIONNALITÉS DE NŒUDS

L'opérateur NFD (Node Feature Discovery) orchestre toutes les ressources nécessaires à l'exécution de l'ensemble de démons NFD. En tant qu'administrateur de cluster, vous pouvez installer l'opérateur NFD à l'aide de la CLI d'OpenShift Container Platform ou de la console Web.

3.2.1. Installation de l'opérateur NFD à l'aide du CLI

En tant qu'administrateur de cluster, vous pouvez installer l'opérateur NFD à l'aide du CLI.

Conditions préalables

- Un cluster OpenShift Container Platform
- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créer un espace de noms pour l'opérateur BDNF.
   a. Créez la ressource personnalisée (CR) **Namespace** suivante qui définit l'espace de noms **openshift-nfd**, puis enregistrez le YAML dans le fichier **nfd-namespace.yaml**:

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

   b. Créez l'espace de noms en exécutant la commande suivante :

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

2. Installez l'opérateur NFD dans l'espace de noms que vous avez créé à l'étape précédente en créant les objets suivants :

   a. Créez le CR **OperatorGroup** suivant et enregistrez le YAML dans le fichier **nfd-operatorgroup.yaml**:

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

   b. Créez le CR **OperatorGroup** en exécutant la commande suivante :

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

   c. Créez le CR **Subscription** suivant et enregistrez le YAML dans le fichier **nfd-sub.yaml**:

   #### Exemple d'abonnement

```
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. Créez l'objet d'abonnement en exécutant la commande suivante :

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

   e. Modification du projet **openshift-nfd**:

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

Vérification

- Pour vérifier que le déploiement de l'opérateur s'est déroulé correctement, exécutez le programme :

```
$ oc get pods
```

Exemple de sortie

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

Un déploiement réussi affiche un statut **Running**.

3.2.2. Installation de l'opérateur NFD à l'aide de la console web

En tant qu'administrateur de cluster, vous pouvez installer l'opérateur NFD à l'aide de la console web.

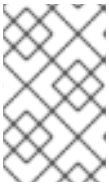
Procédure

1. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
2. Choisissez **Node Feature Discovery** dans la liste des opérateurs disponibles, puis cliquez sur **Install**.
3. Sur la page **Install Operator**, sélectionnez **A specific namespace on the cluster**, puis cliquez sur **Install**. Vous ne devez pas créer d'espace de noms car il est créé pour vous.

Vérification

Pour vérifier que l'opérateur NFD a été installé avec succès :

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Assurez-vous que **Node Feature Discovery** est listé dans le projet **openshift-nfd** avec un **Status** de **InstallSucceeded**.



NOTE

Pendant l'installation, un opérateur peut afficher un état **Failed**. Si l'installation réussit par la suite avec un message **InstallSucceeded**, vous pouvez ignorer le message **Failed**.

Résolution de problèmes

Si l'opérateur n'apparaît pas tel qu'il a été installé, poursuivre le dépannage :

1. Naviguez jusqu'à la page **Operators → Installed Operators** et inspectez les onglets **Operator Subscriptions** et **Install Plans** pour voir s'il n'y a pas de défaillance ou d'erreur sous **Status**.
2. Naviguez vers la page **Workloads → Pods** et vérifiez les journaux pour les pods dans le projet **openshift-nfd**.

3.3. UTILISATION DE L'OPÉRATEUR DE DÉCOUVERTE DE CARACTÉRISTIQUES DE NŒUDS

L'opérateur NFD (Node Feature Discovery) orchestre toutes les ressources nécessaires à l'exécution de l'ensemble de démons Node-Feature-Discovery en surveillant l'arrivée d'un CR **NodeFeatureDiscovery**. Sur la base de la CR **NodeFeatureDiscovery**, l'opérateur crée les composants de l'opérande (BDNF) dans l'espace de noms souhaité. Vous pouvez modifier le CR pour choisir un autre **namespace**, **image**, **imagePullPolicy**, et **nfd-worker-conf**, entre autres options.

En tant qu'administrateur de cluster, vous pouvez créer une instance **NodeFeatureDiscovery** à l'aide du CLI d'OpenShift Container Platform ou de la console Web.

3.3.1. Créer une instance de NodeFeatureDiscovery à l'aide de l'interface de programmation (CLI)

En tant qu'administrateur de cluster, vous pouvez créer une instance **NodeFeatureDiscovery** CR à l'aide de la CLI.

Conditions préalables

- Un cluster OpenShift Container Platform
- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.
- Installer l'opérateur NFD.

Procédure

1. Créez la ressource personnalisée (CR) **NodeFeatureDiscovery** suivante, puis enregistrez le fichier YAML dans le fichier **NodeFeatureDiscovery.yaml**:

```
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"]
```

Pour plus de détails sur la manière de personnaliser les travailleurs NFD, reportez-vous à la référence du fichier de configuration de nfd-worker.

1. Créez l'instance **NodeFeatureDiscovery** CR en exécutant la commande suivante :

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

Vérification

- Pour vérifier que l'instance est créée, exécutez :

```
$ oc get pods
```

Exemple de sortie

```
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
```

Un déploiement réussi affiche un statut **Running**.

3.3.2. Créer un CR NodeFeatureDiscovery à l'aide de la console web

Procédure

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Trouvez **Node Feature Discovery** et voyez une boîte sous **Provided APIs**.
3. Cliquez sur **Create instance**.
4. Modifier les valeurs de **NodeFeatureDiscovery** CR.
5. Cliquez sur **Create**.

3.4. CONFIGURATION DE L'OPÉRATEUR DE DÉCOUVERTE DES FONCTIONNALITÉS DU NŒUD

3.4.1. noyau

La section **core** contient des paramètres de configuration communs qui ne sont pas spécifiques à une source de fonctionnalités particulière.

core.sleepInterval
core.sleepInterval spécifie l'intervalle entre deux passages consécutifs de détection ou de redétection de caractéristiques, et donc également l'intervalle entre deux réétiquetages de nœuds. Une valeur non positive implique un intervalle de sommeil infini ; aucune redétection ou réétiquetage n'est effectué.

Cette valeur est remplacée par l'indicateur de ligne de commande **--sleep-interval**, si elle est spécifiée.

Exemple d'utilisation

```
core:
  sleepInterval: 60s (1)
```

La valeur par défaut est **60s**.

core.sources

core.sources spécifie la liste des sources de fonctionnalités activées. La valeur spéciale **all** permet d'activer toutes les sources de fonctionnalités.

Cette valeur est remplacée par l'indicateur de ligne de commande **--sources**, si elle est spécifiée.

Par défaut : **[all]**

Exemple d'utilisation

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

core.labelWhiteList
core.labelWhiteList spécifie une expression régulière pour filtrer les étiquettes de caractéristiques sur la base du nom de l'étiquette. Les étiquettes qui ne correspondent pas ne sont pas publiées.

L'expression régulière n'est comparée qu'à la partie du nom de base de l'étiquette, c'est-à-dire la partie du nom située après "/". Le préfixe de l'étiquette, ou l'espace de noms, est omis.

Cette valeur est remplacée par l'indicateur de ligne de commande **--label-whitelist**, si elle est spécifiée.

Par défaut : **null**

Exemple d'utilisation

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

core.noPublish
Le réglage de **core.noPublish** sur **true** désactive toute communication avec **nfd-master**. Il s'agit en fait d'un drapeau de marche à vide ; **nfd-worker** exécute la détection des caractéristiques normalement, mais aucune demande d'étiquetage n'est envoyée à **nfd-master**.

Cette valeur est remplacée par l'indicateur de ligne de commande **--no-publish**, s'il est spécifié.

Exemple :

Exemple d'utilisation

```
core:
  noPublish: true (1)
```

La valeur par défaut est **false**.

core.klog
Les options suivantes spécifient la configuration de l'enregistreur, dont la plupart peuvent être ajustées dynamiquement lors de l'exécution.

Les options de l'enregistreur peuvent également être spécifiées à l'aide de drapeaux de ligne de commande, qui ont la priorité sur les options correspondantes du fichier de configuration.

core.klog.addDirHeader

Si la valeur est **true**, **core.klog.addDirHeader** ajoute le répertoire du fichier à l'en-tête des messages d'enregistrement.

Par défaut : **false**

Configurable en cours d'exécution : oui

core.klog.alsologtostderr
Enregistrement dans l'erreur standard et dans les fichiers.

Par défaut : **false**

Configurable en cours d'exécution : oui

core.klog.logBacktraceAt
Lorsque l'enregistrement atteint la ligne file:N, il émet une trace de pile.

Par défaut : **empty**

Configurable en cours d'exécution : oui

core.klog.logDir
S'il n'est pas vide, les fichiers journaux sont écrits dans ce répertoire.

Par défaut : **empty**

Configurable en cours d'exécution : non

core.klog.logFile
S'il n'est pas vide, utilisez ce fichier journal.

Par défaut : **empty**

Configurable en cours d'exécution : non

core.klog.logFileMaxSize
core.klog.logFileMaxSize définit la taille maximale d'un fichier journal. L'unité est le mégaoctet. Si la valeur est **0**, la taille maximale du fichier est illimitée.

Par défaut : **1800**

Configurable en cours d'exécution : non

core.klog.logtostderr
Enregistrer les données dans l'erreur standard au lieu des fichiers

Par défaut : **true**

Configurable en cours d'exécution : oui

core.klog.skipHeaders
Si **core.klog.skipHeaders** est défini sur **true**, évitez les préfixes d'en-tête dans les messages d'information.

Par défaut : **false**

Configurable en cours d'exécution : oui

core.klog.skipLogHeaders
Si **core.klog.skipLogHeaders** est défini sur **true**, éviter les en-têtes lors de l'ouverture des fichiers journaux.

Par défaut : **false**

Configurable en cours d'exécution : non

core.klog.stderrthreshold
Les journaux qui atteignent ou dépassent ce seuil sont envoyés à stderr.

Par défaut : **2**

Configurable en cours d'exécution : oui

core.klog.v
core.klog.v est le numéro du niveau de verbosité du journal.

Par défaut : **0**

Configurable en cours d'exécution : oui

core.klog.vmodule
core.klog.vmodule est une liste de paramètres **pattern=N** séparés par des virgules pour la journalisation avec filtrage des fichiers.

Par défaut : **empty**

Configurable en cours d'exécution : oui

3.4.2. sources

La section **sources** contient des paramètres de configuration spécifiques aux sources de fonctionnalités.

sources.cpu.cpuid.attributeBlacklist
Empêcher la publication de **cpuid** des fonctionnalités énumérées dans cette option.

Cette valeur est remplacée par **sources.cpu.cpuid.attributeWhitelist**, si elle est spécifiée.

Par défaut : **[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]**

Exemple d'utilisation

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

sources.cpu.cpuid.attributeWhitelist
Publier uniquement les fonctionnalités de **cpuid** énumérées dans cette option.

sources.cpu.cpuid.attributeWhitelist a la priorité sur **sources.cpu.cpuid.attributeBlacklist**.

Par défaut : **empty**

Exemple d'utilisation

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

sources.kernel.kconfigFile
sources.kernel.kconfigFile est le chemin du fichier de configuration du noyau. S'il est vide, NFD effectue une recherche dans les emplacements standard connus.

Par défaut : **empty**

Exemple d'utilisation

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

sources.kernel.configOpts
sources.kernel.configOpts représente les options de configuration du noyau à publier sous forme d'étiquettes d'éléments.

Par défaut : **[NO_HZ, NO_HZ_IDLE, NO_HZ_FULL, PREEMPT]**

Exemple d'utilisation

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

sources.pci.deviceClassWhitelist
sources.pci.deviceClassWhitelist est une liste d'identifiants de classe de périphériques PCI pour lesquels il faut publier une étiquette. Elle peut être spécifiée en tant que classe principale uniquement (par exemple, **03**) ou en tant que combinaison complète de classe et de sous-classe (par exemple, **0300**). Dans le premier cas, toutes les sous-classes sont acceptées. Le format des étiquettes peut être configuré de manière plus approfondie à l'aide de **deviceLabelFields**.

Par défaut : **["03", "0b40", "12"]**

Exemple d'utilisation

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

sources.pci.deviceLabelFields
sources.pci.deviceLabelFields est l'ensemble des champs d'identification PCI à utiliser pour construire le nom de l'étiquette de la caractéristique. Les champs valides sont **class**, **vendor**, **device**, **subsystem_vendor** et **subsystem_device**.

Par défaut : **[class, vendor]**

Exemple d'utilisation

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

Avec l'exemple de configuration ci-dessus, NFD publierait des étiquettes telles que **feature.node.kubernetes.io/pci-<class-id>_<vendor-id>_<device-id>.present=true**

sources.usb.deviceClassWhitelist
sources.usb.deviceClassWhitelist est une liste d'identifiants de classe de périphériques USB pour lesquels il convient de publier une étiquette de fonctionnalité. Le format des étiquettes peut être configuré à l'aide de **deviceLabelFields**.

Par défaut : **["0e", "ef", "fe", "ff"]**

Exemple d'utilisation

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

sources.usb.deviceLabelFields
sources.usb.deviceLabelFields est l'ensemble des champs d'identification USB à partir desquels doit être composé le nom de l'étiquette de la fonctionnalité. Les champs valides sont **class**, **vendor** et **device**.

Par défaut : **[class, vendor, device]**

Exemple d'utilisation

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

Avec l'exemple de configuration ci-dessus, NFD publierait des étiquettes comme : **feature.node.kubernetes.io/usb-<class-id>_<vendor-id>.present=true**.

sources.custom
sources.custom est la liste des règles à traiter dans la source de fonctionnalités personnalisées pour créer des étiquettes spécifiques à l'utilisateur.

Par défaut : **empty**

Exemple d'utilisation

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

3.5. UTILISATION DE L'OUTIL DE MISE À JOUR DE LA TOPOLOGIE NFD

L'outil de mise à jour de la topologie NFD (Node Feature Discovery) est un démon chargé d'examiner les ressources allouées sur un nœud de travail. Il comptabilise les ressources disponibles pour être allouées à un nouveau pod sur une base par zone, une zone pouvant être un nœud NUMA (Non-Uniform Memory Access). L'outil NFD Topology Updater communique les informations à nfd-master, qui crée une ressource personnalisée (CR) **NodeResourceTopology** correspondant à tous les nœuds de travail de la grappe. Une instance de NFD Topology Updater s'exécute sur chaque nœud de la grappe.

Pour activer les travailleurs de la mise à jour de la topologie dans la BDNF, réglez la variable **topologyupdater** sur **true** dans la CR **NodeFeatureDiscovery**, comme décrit dans la section **Using the Node Feature Discovery Operator**.

3.5.1. NodeResourceTopology CR

Lorsqu'il est exécuté avec NFD Topology Updater, NFD crée des instances de ressources personnalisées correspondant à la topologie matérielle des ressources du nœud, par exemple :

```
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. Drapeaux de la ligne de commande de l'outil de mise à jour de la topologie NFD

Pour afficher les drapeaux de ligne de commande disponibles, exécutez la commande **nfd-topology-updater -help**. Par exemple, dans un conteneur podman, exécutez la commande suivante :

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

-fichier ca
Le drapeau **-ca-file** est l'un des trois drapeaux, avec les drapeaux **-cert-file** et `-key-file`, qui contrôle l'authentification mutuelle TLS sur le NFD Topology Updater. Ce drapeau spécifie le certificat racine TLS utilisé pour vérifier l'authenticité de nfd-master.

Défaut : vide

IMPORTANT

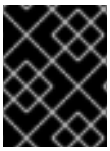
L'indicateur **-ca-file** doit être spécifié en même temps que les indicateurs **-cert-file** et **-key-file**.

Exemple :

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

-fichier-certificat
L'indicateur **-cert-file** est l'un des trois indicateurs, avec les indicateurs **-ca-file** et **-key-file flags**, qui contrôlent l'authentification TLS mutuelle sur l'outil de mise à jour de la topologie NFD. Cet indicateur spécifie le certificat TLS présenté pour l'authentification des requêtes sortantes.

Défaut : vide



IMPORTANT

L'indicateur **-cert-file** doit être spécifié en même temps que les indicateurs **-ca-file** et **-key-file**.

Exemple :

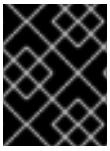
```
$ nfd-topology-updater -cert-file=/opt/nfd/updater.crt -key-file=/opt/nfd/updater.key -ca-
file=/opt/nfd/ca.crt
```

-h, -help
Imprimer l'utilisation et quitter.

-fichier-clé
Le drapeau **-key-file** est l'un des trois drapeaux, avec les drapeaux **-ca-file** et **-cert-file**, qui contrôlent l'authentification mutuelle TLS sur le NFD Topology Updater. Ce drapeau spécifie la clé privée correspondant au fichier de certificat donné, ou **-cert-file**, qui est utilisée pour authentifier les demandes

sortantes.

Défaut : vide



IMPORTANT

L'indicateur **-key-file** doit être spécifié en même temps que les indicateurs **-ca-file** et **-cert-file**.

Exemple :

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

-kubelet-config-file
L'adresse **-kubelet-config-file** indique le chemin d'accès au fichier de configuration de la Kubelet.

Par défaut : **/host-var/lib/kubelet/config.yaml**

Exemple :

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

-pas de publication
Le drapeau **-no-publish** désactive toute communication avec le maître nfd, ce qui en fait un drapeau de fonctionnement à vide pour nfd-topology-updater. NFD Topology Updater exécute normalement la détection de topologie matérielle des ressources, mais aucune demande de CR n'est envoyée à nfd-master.

Par défaut : **false**

Exemple :

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

3.5.2.1. -one-shot

L'indicateur **-oneshot** permet à l'outil de mise à jour de la topologie NFD de quitter le système après un passage de la détection de la topologie matérielle des ressources.

Par défaut : **false**

Exemple :

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

-podresources-socket
L'option **-podresources-socket** spécifie le chemin d'accès au socket Unix où kubelet exporte un service gRPC pour permettre la découverte des unités centrales et des périphériques en cours d'utilisation, et pour fournir des métadonnées à leur sujet.

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

Exemple :

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

-serveur
Le drapeau **-server** spécifie l'adresse du point de terminaison nfd-master auquel se connecter.

Par défaut : **localhost:8080**

Exemple :

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

-surcharge du nom du serveur
L'option **-server-name-override** spécifie le nom commun (CN) que l'on attend du certificat TLS de nfd-master. Cet indicateur est principalement destiné au développement et au débogage.

Défaut : vide

Exemple :

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

-intervalle de sommeil
L'indicateur **-sleep-interval** précise l'intervalle entre le réexamen de la topologie matérielle des ressources et les mises à jour des ressources personnalisées. Une valeur non positive implique un intervalle de sommeil infini et aucune re-détection n'est effectuée.

Par défaut : **60s**

Exemple :

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

-version
Imprimer la version et quitter.

-espace de veille
L'option **-watch-namespace** spécifie l'espace de noms pour s'assurer que l'examen de la topologie matérielle des ressources ne concerne que les modules fonctionnant dans l'espace de noms spécifié. Les modules qui ne sont pas exécutés dans l'espace de noms spécifié ne sont pas pris en compte lors de la comptabilisation des ressources. Cette fonction est particulièrement utile pour les tests et le débogage. Une valeur de * signifie que tous les pods de tous les espaces de noms sont pris en compte lors du processus de comptabilisation.

Par défaut : *

Exemple :

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

CHAPITRE 4. OPÉRATEUR DE GESTION DES MODULES DU NOYAU

Découvrez l'opérateur Kernel Module Management (KMM) et comment vous pouvez l'utiliser pour déployer des modules de noyau hors arbre et des plugins de périphérique sur les clusters OpenShift Container Platform.

4.1. À PROPOS DE L'OPÉRATEUR DE GESTION DES MODULES DU NOYAU

L'opérateur Kernel Module Management (KMM) gère, construit, signe et déploie des modules de noyau hors arbre et des plugins de périphériques sur les clusters OpenShift Container Platform.

KMM ajoute un nouveau CRD **Module** qui décrit un module de noyau hors-arbre et son plugin de périphérique associé. Vous pouvez utiliser les ressources **Module** pour configurer le chargement du module, définir les images **ModuleLoader** pour les versions du noyau et inclure des instructions pour construire et signer des modules pour des versions spécifiques du noyau.

KMM est conçu pour prendre en charge plusieurs versions de noyau à la fois pour n'importe quel module de noyau, ce qui permet des mises à niveau transparentes des nœuds et une réduction des temps d'arrêt des applications.

4.2. INSTALLATION DE L'OPÉRATEUR DE GESTION DU MODULE DU NOYAU

En tant qu'administrateur de cluster, vous pouvez installer l'opérateur Kernel Module Management (KMM) en utilisant la CLI d'OpenShift ou la console web.

L'opérateur KMM est pris en charge sur OpenShift Container Platform 4.12 et plus. L'installation de KMM sur la version 4.11 ne nécessite pas d'étapes supplémentaires spécifiques. Pour plus de détails sur l'installation de KMM sur la version 4.10 et les versions antérieures, voir la section "Installation de l'opérateur de gestion des modules du noyau sur les versions antérieures d'OpenShift Container Platform".

4.2.1. Installation de l'opérateur de gestion du module du noyau à l'aide de la console web

En tant qu'administrateur de cluster, vous pouvez installer l'opérateur Kernel Module Management (KMM) à l'aide de la console web d'OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Installez l'opérateur de gestion du module du noyau :
    a. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
    b. Sélectionnez **Kernel Module Management Operator** dans la liste des opérateurs disponibles, puis cliquez sur **Install**.

c. Sur la page **Install Operator**, sélectionnez le site **Installation mode** comme **A specific namespace on the cluster**.

d. Dans la liste **Installed Namespace**, sélectionnez l'espace de noms **openshift-kmm**.

e. Cliquez sur **Install**.

Vérification

Pour vérifier que l'installation de KMM Operator s'est déroulée correctement :

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Assurez-vous que **Kernel Module Management Operator** est listé dans le projet **openshift-kmm** avec un **Status** de **InstallSucceeded**.

NOTE

Pendant l'installation, un opérateur peut afficher un état **Failed**. Si l'installation réussit par la suite avec un message **InstallSucceeded**, vous pouvez ignorer le message **Failed**.

Résolution de problèmes

1. Pour résoudre les problèmes liés à l'installation de l'opérateur :
   a. Naviguez jusqu'à la page **Operators → Installed Operators** et inspectez les onglets **Operator Subscriptions** et **Install Plans** pour voir s'il n'y a pas de défaillance ou d'erreur sous **Status**.
   b. Naviguez vers la page **Workloads → Pods** et vérifiez les journaux pour les pods dans le projet **openshift-kmm**.

4.2.2. Installation de l'opérateur de gestion du module du noyau à l'aide de la CLI

En tant qu'administrateur de cluster, vous pouvez installer l'opérateur Kernel Module Management (KMM) à l'aide de la CLI d'OpenShift.

Conditions préalables

- Vous disposez d'un cluster OpenShift Container Platform en cours d'exécution.
- You installed the OpenShift CLI (**oc**).
- Vous êtes connecté à OpenShift CLI en tant qu'utilisateur avec des privilèges **cluster-admin**.

Procédure

1. Installer KMM dans l'espace de noms **openshift-kmm**:
   a. Créez le fichier **Namespace** CR suivant et enregistrez le fichier YAML, par exemple, **kmm-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
```

```
    name: openshift-kmm
```

b. Créez le fichier **OperatorGroup** CR suivant et enregistrez le fichier YAML, par exemple, **kmm-op-group.yaml**:

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

c. Créez le fichier **Subscription** CR suivant et enregistrez le fichier YAML, par exemple, **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. Créez l'objet d'abonnement en exécutant la commande suivante :

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

Vérification

- Pour vérifier que le déploiement de l'opérateur a réussi, exécutez la commande suivante :

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

Exemple de sortie

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

L'opérateur est disponible.

4.2.3. Installation de l'opérateur de gestion des modules du noyau sur les versions antérieures d'OpenShift Container Platform

L'opérateur KMM est pris en charge sur OpenShift Container Platform 4.12 et plus. Pour les versions 4.10 et antérieures, vous devez créer un nouvel objet **SecurityContextConstraint** et le lier au site **ServiceAccount** de l'opérateur. En tant qu'administrateur de cluster, vous pouvez installer l'opérateur Kernel Module Management (KMM) à l'aide de la CLI OpenShift.

Conditions préalables

- Vous disposez d'un cluster OpenShift Container Platform en cours d'exécution.
- You installed the OpenShift CLI (**oc**).
- Vous êtes connecté à OpenShift CLI en tant qu'utilisateur avec des privilèges **cluster-admin**.

Procédure

1. Installer KMM dans l'espace de noms **openshift-kmm**:
   a. Créez le CR **Namespace** suivant et enregistrez le fichier YAML, par exemple, le fichier **kmm-namespace.yaml**:

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

   b. Créez l'objet **SecurityContextConstraint** suivant et enregistrez le fichier YAML, par exemple **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. Liez l'objet **SecurityContextConstraint** à l'objet **ServiceAccount** de l'opérateur en exécutant les commandes suivantes :

```
$ 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. Créez le fichier **OperatorGroup** CR suivant et enregistrez le fichier YAML, par exemple, **kmm-op-group.yaml**:

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

e. Créez le fichier **Subscription** CR suivant et enregistrez le fichier YAML, par exemple, **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. Créez l'objet d'abonnement en exécutant la commande suivante :

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

Vérification

- Pour vérifier que le déploiement de l'opérateur a réussi, exécutez la commande suivante :

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

Exemple de sortie

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

L'opérateur est disponible.

4.3. DÉPLOIEMENT DU MODULE DU NOYAU

Pour chaque ressource **Module**, Kernel Module Management (KMM) peut créer un certain nombre de ressources **DaemonSet**:

- Un ModuleLoader **DaemonSet** par version de noyau compatible fonctionnant dans le cluster.
- Un plugin d'appareil **DaemonSet**, s'il est configuré.

Les ressources du démon de chargement de modules exécutent des images ModuleLoader pour charger les modules du noyau. Une image de chargeur de modules est une image OCI qui contient les fichiers **.ko** et les binaires **modprobe** et **sleep**.

Lorsque le module loader pod est créé, il exécute **modprobe** pour insérer le module spécifié dans le noyau. Il entre ensuite dans un état de veille jusqu'à ce qu'il soit terminé. À ce moment-là, le crochet **ExecPreStop** exécute **modprobe -r** pour décharger le module du noyau.

Si l'attribut **.spec.devicePlugin** est configuré dans une ressource **Module**, KMM crée un ensemble de démons de plugins de périphériques dans le cluster. Ce jeu de démons cible :

- Nœuds correspondant au site **.spec.selector** de la ressource **Module**.
- Nœuds avec le module du noyau chargé (où le module loader pod est dans l'état **Ready**).

4.3.1. La définition de la ressource personnalisée du module

La définition de ressource personnalisée (CRD) **Module** représente un module de noyau qui peut être chargé sur tous les nœuds du cluster ou sur certains d'entre eux, par l'intermédiaire d'une image de chargeur de module. Une ressource personnalisée **Module** (CR) spécifie une ou plusieurs versions du noyau avec lesquelles elle est compatible, ainsi qu'un sélecteur de nœud.

Les versions compatibles d'une ressource **Module** sont répertoriées sous **.spec.moduleLoader.container.kernelMappings**. Un mappage de noyau peut soit correspondre à une version **literal**, soit utiliser **regexp** pour correspondre à plusieurs d'entre elles en même temps.

La boucle de rapprochement pour la ressource **Module** se déroule comme suit :

1. Liste de tous les nœuds correspondant à **.spec.selector**.
2. Construire un ensemble de toutes les versions du noyau fonctionnant sur ces nœuds.
3. Pour chaque version du noyau :
    a. Parcourez **.spec.moduleLoader.container.kernelMappings** et trouvez le nom de l'image de conteneur appropriée. Si le mappage du noyau est défini par **build** ou **sign** et que l'image du conteneur n'existe pas encore, exécutez la compilation, le travail de signature, ou les deux, selon les besoins.
    b. Créez un daemon de chargement de modules avec l'image du conteneur déterminée à l'étape précédente.
    c. Si **.spec.devicePlugin** est défini, créez un ensemble de démons d'extension de périphérique en utilisant la configuration spécifiée sous **.spec.devicePlugin.container**.

4. Lancer **garbage-collect** sur :

    a. Les ressources du jeu de démons existant ciblent les versions du noyau qui ne sont exécutées par aucun nœud de la grappe.

    b. Des emplois réussis dans le domaine de la construction.

    c. Des signatures réussies.

4.3.2. Sécurité et autorisations

IMPORTANT

Le chargement des modules du noyau est une opération très sensible. Une fois chargés, les modules du noyau disposent de toutes les autorisations possibles pour effectuer n'importe quel type d'opération sur le nœud.

4.3.2.1. ServiceAccounts et SecurityContextConstraints

Kernel Module Management (KMM) crée une charge de travail privilégiée pour charger les modules du noyau sur les nœuds. Cette charge de travail doit être **ServiceAccounts** autorisée à utiliser la ressource **privileged SecurityContextConstraint** (SCC).

Le modèle d'autorisation pour cette charge de travail dépend de l'espace de noms de la ressource **Module**, ainsi que de sa spécification.

- Si les champs **.spec.moduleLoader.serviceAccountName** ou **.spec.devicePlugin.serviceAccountName** sont définis, ils sont toujours utilisés.
- Si ces champs ne sont pas renseignés, alors :
    - Si la ressource **Module** est créée dans l'espace de noms de l'opérateur (**openshift-kmm** par défaut), KMM utilise alors sa ressource par défaut, **ServiceAccounts**, pour exécuter les ensembles de démons.
    - Si la ressource **Module** est créée dans un autre espace de noms, KMM exécute les ensembles de démons en tant que **default ServiceAccount** de l'espace de noms. La ressource **Module** ne peut pas exécuter une charge de travail privilégiée à moins que vous ne l'autorisiez manuellement à utiliser le SCC **privileged**.

IMPORTANT

openshift-kmm est un espace de noms de confiance.

Lors de la configuration des autorisations RBAC, n'oubliez pas que tout utilisateur ou **ServiceAccount** créant une ressource **Module** dans l'espace de noms **openshift-kmm** entraîne l'exécution automatique par KMM de charges de travail privilégiées sur potentiellement tous les nœuds de la grappe.

Pour permettre à n'importe quel site **ServiceAccount** d'utiliser le SCC **privileged** et donc d'exécuter des pods de chargement de modules ou de plugins de périphériques, utilisez la commande suivante :

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

4.3.2.2. Normes de sécurité des pods

OpenShift exécute un mécanisme de synchronisation qui définit automatiquement le niveau de sécurité de l'espace de noms Pod Security en fonction des contextes de sécurité utilisés. Aucune action n'est nécessaire.

Ressources supplémentaires

- Comprendre et gérer l'admission à la sécurité des pods .

4.3.3. Exemple de module CR

Voici un exemple de **Module** annoté :

```
apiVersion: kmm.sigs.x-k8s.io/v1beta1
kind: Module
metadata:
  name: <my_kmod>
spec:
  moduleLoader:
    container:
      modprobe:
        moduleName: <my_kmod> 1
        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: '^.+$' 7
          containerImage: "some.registry/org/<my_kmod>:${KERNEL_FULL_VERSION}"
          build:
            buildArgs: 8
              - name: ARG_NAME
                value: <some_value>
            secrets:
              - name: <some_kubernetes_secret> 9
            baseImageRegistryTLS: 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 Required.

2 En option.

3 Facultatif : Copie **/firmware/*** dans **/var/lib/firmware/** sur le nœud.

4 En option.

5 Au moins un élément du noyau est requis.

6 Pour chaque nœud exécutant un noyau correspondant à l'expression régulière, KMM crée une ressource **DaemonSet** exécutant l'image spécifiée dans **containerImage**, **${KERNEL_FULL_VERSION}** étant remplacé par la version du noyau.

7 Pour tout autre noyau, construisez l'image en utilisant le fichier Docker dans le ConfigMap de **my-kmod**.

8 En option.

9 Facultatif : Une valeur pour **some-kubernetes-secret** peut être obtenue à partir de l'environnement de construction à l'adresse **/run/secrets/some-kubernetes-secret**.

10 Facultatif : Évitez d'utiliser ce paramètre. Si ce paramètre vaut **true**, la compilation est autorisée à extraire l'image dans l'instruction Dockerfile **FROM** en utilisant le protocole HTTP.

11 Facultatif : Évitez d'utiliser ce paramètre. S'il vaut **true**, la compilation ignorera toute validation de certificat de serveur TLS lors de l'extraction de l'image dans l'instruction Dockerfile **FROM** à l'aide d'un simple HTTP.

12 Required.

13 Nécessaire : Un secret contenant la clé publique secureboot avec la clé "cert".

14 Nécessaire : Un secret contenant la clé privée de démarrage sécurisé avec la clé "key".

15 Facultatif : Évitez d'utiliser ce paramètre. S'il vaut **true**, KMM sera autorisé à vérifier si l'image du conteneur existe déjà en utilisant le protocole HTTP ordinaire.

16 Facultatif : Évitez d'utiliser ce paramètre. S'il est défini sur **true**, KMM ignorera toute validation de certificat de serveur TLS lors de la vérification de l'existence de l'image du conteneur.

17 En option.

18 En option.

19 Obligatoire : Si la section plugin de l'appareil est présente.

20 En option.

21 En option.

22 En option.

23 Facultatif : Utilisé pour extraire les images du chargeur de module et du plugin de périphérique.

4.4. UTILISATION D'UNE IMAGE MODULELOADER

La gestion des modules du noyau (KMM) fonctionne avec des images de chargeurs de modules spécifiques. Il s'agit d'images OCI standard qui doivent satisfaire aux exigences suivantes :

- **.ko** les fichiers doivent être situés à l'adresse **/opt/lib/modules/${KERNEL_VERSION}**.
- **modprobe** et **sleep** doivent être définis dans la variable **$PATH**.

4.4.1. Exécution de depmod

Si votre image de chargeur de modules contient plusieurs modules du noyau et si l'un des modules dépend d'un autre module, il est préférable d'exécuter **depmod** à la fin du processus de construction pour générer les dépendances et les fichiers de mappage.

NOTE

Vous devez avoir un abonnement Red Hat pour télécharger le paquetage **kernel-devel**.

Procédure

1. Pour générer les fichiers **modules.dep** et **.map** pour une version spécifique du noyau, exécutez **depmod -b /opt ${KERNEL_VERSION}**.

4.4.1.1. Example Dockerfile

Si vous construisez votre image sur OpenShift Container Platform, pensez à utiliser le Driver Tool Kit (DTK).

Pour plus d'informations, voir l'utilisation d'un build intitulé.

```
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}
```

Ressources supplémentaires

- Driver Toolkit.

4.4.2. Construire dans le cluster

KMM peut construire des images de chargeurs de modules dans le cluster. Suivez les instructions suivantes :

- Fournir des instructions de compilation en utilisant la section **build** d'un mappage de noyau.
- Copiez l'adresse **Dockerfile** de votre image de conteneur dans une ressource **ConfigMap**, sous la clé **dockerfile**.
- Assurez-vous que le site **ConfigMap** se trouve dans le même espace de noms que le site **Module**.

KMM vérifie si le nom de l'image spécifié dans le champ **containerImage** existe. Si c'est le cas, la construction est ignorée.

Sinon, KMM crée une ressource **Build** pour construire votre image. Une fois l'image construite, KMM procède au rapprochement avec **Module**. Voir l'exemple suivant.

```
# ...
- 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
```

1. En option.
2. En option.
3. Il sera monté dans le module de construction comme **/run/secrets/some-kubernetes-secret**.
4. Facultatif : Évitez d'utiliser ce paramètre. Si ce paramètre vaut **true**, la compilation sera autorisée à extraire l'image dans l'instruction Dockerfile **FROM** en utilisant le protocole HTTP.
5. Facultatif : Évitez d'utiliser ce paramètre. S'il vaut **true**, la compilation ignorera toute validation de certificat de serveur TLS lors de l'extraction de l'image dans l'instruction Dockerfile **FROM** à l'aide d'un simple HTTP.
6. Required.
7. Facultatif : Évitez d'utiliser ce paramètre. S'il vaut **true**, KMM sera autorisé à vérifier si l'image du conteneur existe déjà en utilisant le protocole HTTP ordinaire.
8. Facultatif : Évitez d'utiliser ce paramètre. S'il est défini sur **true**, KMM ignorera toute validation de certificat de serveur TLS lors de la vérification de l'existence de l'image du conteneur.

Ressources supplémentaires

- Construire des ressources de configuration .

4.4.3. Utilisation du Driver Toolkit

Le Driver Toolkit (DTK) est une image de base pratique pour construire des images de chargeur de module de construction. Il contient des outils et des bibliothèques pour la version d'OpenShift en cours d'exécution dans le cluster.

Procédure

Utiliser le DTK comme première étape d'un processus en plusieurs étapes **Dockerfile**.

1. Construire les modules du noyau.
2. Copiez les fichiers **.ko** dans une image plus petite destinée à l'utilisateur final, telle que **ubi-minimal**.
3. Pour tirer parti de DTK dans votre compilation en grappe, utilisez l'argument de compilation **DTK_AUTO**. La valeur est automatiquement définie par KMM lors de la création de la ressource **Build**. Voir l'exemple suivant.

```
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}
```

Ressources supplémentaires

- Driver Toolkit.

4.5. UTILISATION DE LA SIGNATURE AVEC KERNEL MODULE MANAGEMENT (KMM)

Sur un système compatible avec Secure Boot, tous les modules du noyau (kmods) doivent être signés à l'aide d'une paire de clés publique/privée enregistrée dans la base de données MOK (Machine Owner's Key). Les pilotes distribués dans le cadre d'une distribution doivent déjà être signés par la clé privée de la distribution, mais pour les modules du noyau construits en dehors de l'arbre, KMM prend en charge la signature des modules du noyau à l'aide de la section **sign** du mappage du noyau.

Pour plus de détails sur l'utilisation de Secure Boot, voir Générer une paire de clés publique et privée

Conditions préalables

- Une paire de clés publiques et privées dans le format correct (DER).
- Au moins un nœud à démarrage sécurisé dont la clé publique est enregistrée dans sa base de données MOK.
- Soit une image de conteneur de pilote préconstruite, soit le code source et le site **Dockerfile** nécessaires à la construction d'un conteneur en cluster.

4.6. AJOUT DES CLÉS POUR SECUREBOOT

Pour utiliser KMM Kernel Module Management (KMM) afin de signer les modules du noyau, un certificat et une clé privée sont nécessaires. Pour savoir comment les créer, voir Générer une paire de clés publique et privée.

Pour plus d'informations sur l'extraction de la paire de clés publique et privée, voir Signer les modules du noyau avec la clé privée. Utilisez les étapes 1 à 4 pour extraire les clés dans des fichiers.

Procédure

1. Créez le fichier **sb_cert.cer** qui contient le certificat et le fichier **sb_cert.priv** qui contient la clé privée :

```
$ 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. Ajoutez les fichiers en utilisant l'une des méthodes suivantes :

- Ajouter directement les fichiers en tant que secrets:

```
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>
```

- Ajouter les fichiers en les encodant en base64 :

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

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

3. Ajouter le texte encodé à un fichier YAML :

```
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** - Remplacer **default** par un espace de noms valide.

4. Appliquer le fichier YAML :

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

4.6.1. Vérification des clés

Après avoir ajouté les clés, vous devez les vérifier pour vous assurer qu'elles sont correctement réglées.

Procédure

1. Vérifiez que le secret de la clé publique est correctement défini :

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

Le certificat devrait s'afficher avec un numéro de série, un émetteur, un sujet, etc.

2. Vérifiez que le secret de la clé privée est correctement défini :

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

Cela devrait permettre d'afficher la clé contenue dans les lignes **-----BEGIN PRIVATE KEY-----** et **-----END PRIVATE KEY-----**.

4.7. SIGNER UN CONTENEUR DE PILOTE PRÉCONSTRUIT

Utilisez cette procédure si vous disposez d'une image préconstruite, telle qu'une image distribuée par un fournisseur de matériel ou construite ailleurs.

Le fichier YAML suivant ajoute la paire de clés publique/privée en tant que secrets avec les noms de clés requis - **key** pour la clé privée, **cert** pour la clé publique. Le cluster extrait ensuite l'image **unsignedImage**, l'ouvre, signe les modules du noyau répertoriés dans **filesToSign**, les réintègre et pousse l'image résultante sous **containerImage**.

Kernel Module Management (KMM) devrait alors déployer le DaemonSet qui charge les kmods signés sur tous les nœuds qui correspondent au sélecteur. Les conteneurs de pilotes doivent s'exécuter avec succès sur tous les nœuds dont la base de données MOK contient la clé publique, ainsi que sur tous les nœuds dont le démarrage sécurisé n'est pas activé, qui ignorent la signature. Ils ne doivent pas se charger sur les nœuds dont le démarrage sécurisé est activé, mais dont la base de données MOK ne contient pas la clé.

Conditions préalables

- Les secrets **keySecret** et **certSecret** ont été créés.

Procédure

1. Appliquer le fichier YAML :

```
---
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** - Le nom du kmod à charger.

4.8. CONSTRUCTION ET SIGNATURE D'UNE IMAGE DE CONTENEUR MODULELOADER

Utilisez cette procédure si vous disposez d'un code source et que vous devez d'abord créer votre image.

Le fichier YAML suivant construit une nouvelle image de conteneur en utilisant le code source du référentiel. L'image produite est sauvegardée dans le registre avec un nom temporaire, et cette image temporaire est ensuite signée en utilisant les paramètres de la section **sign**.

Le nom de l'image temporaire est basé sur le nom de l'image finale et est fixé à **<containerImage>:<tag>-<namespace>_<module name>_kmm_unsigned**.

Par exemple, en utilisant le fichier YAML suivant, Kernel Module Management (KMM) construit une image nommée **example.org/repository/minimal-driver:final-default_example-module_kmm_unsigned** contenant la compilation avec des kmods non signés et l'envoie au registre. Il crée ensuite une deuxième image nommée **example.org/repository/minimal-driver:final** qui contient les kmods signés. C'est cette seconde image qui est chargée par l'objet **DaemonSet** et qui déploie les kmods sur les nœuds du cluster.

Une fois signée, l'image temporaire peut être supprimée du registre en toute sécurité. Elle sera reconstruite si nécessaire.

Conditions préalables

- Les secrets **keySecret** et **certSecret** ont été créés.

Procédure

1. Appliquer le fichier YAML :

```
---
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 (2)
spec:
  moduleLoader:
    serviceAccountName: default (3)
    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: (4)
    name: repo-pull-secret
  selector: # top-level selector
    kubernetes.io/arch: amd64
```

(1) (2) **namespace** - Remplacer **default** par un espace de noms valide.

(3) **serviceAccountName** - Le compte par défaut **serviceAccountName** ne dispose pas des autorisations nécessaires pour exécuter un module privilégié. Pour plus d'informations sur la création d'un compte de service, voir "Création de comptes de service" dans le paragraphe "Ressources supplémentaires" de cette section.

(4) **imageRepoSecret** - Utilisé comme **imagePullSecrets** dans l'objet **DaemonSet** et pour tirer et pousser pour les fonctions de construction et de signature.

Ressources supplémentaires

Pour plus d'informations sur la création d'un compte de service, voir Création de comptes de service .

4.9. DÉBOGAGE ET DÉPANNAGE

Si les kmods de votre conteneur de pilotes ne sont pas signés ou sont signés avec la mauvaise clé, le conteneur peut entrer dans un état **PostStartHookError** ou **CrashLoopBackOff**. Vous pouvez le vérifier en exécutant la commande **oc describe** sur votre conteneur, qui affiche le message suivant dans ce scénario :

```
modprobe : ERROR : Impossible d'insérer '<votre_nom_de_kmod>' : La clé requise n'est pas disponible
```

4.10. PRISE EN CHARGE DU MICROLOGICIEL KMM

Les modules du noyau doivent parfois charger des fichiers de microprogrammes à partir du système de fichiers. KMM prend en charge la copie des fichiers de microprogrammes de l'image ModuleLoader vers le système de fichiers du nœud.

Le contenu de **.spec.moduleLoader.container.modprobe.firmwarePath** est copié dans le chemin **/var/lib/firmware** sur le nœud avant d'exécuter la commande **modprobe** pour insérer le module du noyau.

Tous les fichiers et répertoires vides sont supprimés de cet emplacement avant d'exécuter la commande **modprobe -r** pour décharger le module du noyau, lorsque le pod est terminé.

Ressources supplémentaires

- Création d'une image ModuleLoader.

4.10.1. Configuration du chemin de recherche sur les nœuds

Sur les nœuds d'OpenShift Container Platform, l'ensemble des chemins de recherche par défaut pour les firmwares n'inclut pas le chemin **/var/lib/firmware**.

Procédure

1. Utilisez l'opérateur Machine Config pour créer une ressource personnalisée (CR) **MachineConfig** qui contient le chemin d'accès **/var/lib/firmware**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 99-worker-kernel-args-firmware-path
spec:
  kernelArguments:
    - 'firmware_class.path=/var/lib/firmware'
```

(1) Vous pouvez configurer l'étiquette en fonction de vos besoins. Dans le cas d'OpenShift à nœud unique, utilisez les objets **control-pane** ou **master**.

2. En appliquant le CR **MachineConfig**, les nœuds sont automatiquement redémarrés.

Ressources supplémentaires

- Opérateur Config Machine.

4.10.2. Construction d'une image ModuleLoader

Procédure

- En plus de construire le module du noyau lui-même, il faut inclure le micrologiciel binaire dans l'image du constructeur :

```
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. Optimisation des ressources du module

Procédure

- Définir **.spec.moduleLoader.container.modprobe.firmwarePath** dans la ressource personnalisée (CR) **Module**:

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

        firmwarePath: /firmware (1)
```

(1) Facultatif : Copie **/firmware/*** dans **/var/lib/firmware/** sur le nœud.

4.11. DÉPANNAGE DE KMM

Lors du dépannage des problèmes d'installation de KMM, vous pouvez surveiller les journaux afin de déterminer à quelle étape les problèmes surviennent. Vous pouvez ensuite récupérer les données de diagnostic correspondant à cette étape.

4.11.1. Utilisation de l'outil must-gather

La commande **oc adm must-gather** est la méthode préférée pour collecter un paquet d'assistance et fournir des informations de débogage à l'assistance de Red Hat. Collectez des informations spécifiques en exécutant la commande avec les arguments appropriés, comme décrit dans les sections suivantes.

Ressources supplémentaires

- À propos de l'outil de collecte obligatoire

4.11.1.1. Collecte de données pour la KMM

Procédure

1. Rassembler les données pour le gestionnaire du contrôleur de l'opérateur KMM :

   a. Définir la variable **MUST_GATHER_IMAGE**:

   ```
   $ export MUST_GATHER_IMAGE=$(oc get deployment -n openshift-kmm kmm-operator-controller-manager -ojsonpath='{.spec.template.spec.containers[?(@.name=="manager")].env[?(@.name=="RELATED_IMAGES_MUST_GATHER")].value}')
   ```

   NOTE

   Utilisez le commutateur **-n <namespace>** pour spécifier un espace de noms si vous avez installé KMM dans un espace de noms personnalisé.

   b. Exécutez l'outil **must-gather**:

   ```
   $ oc adm must-gather --image="${MUST_GATHER_IMAGE}" -- /usr/bin/gather
   ```

2. Consulter les journaux de l'opérateur :

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

   Exemple 4.1. Exemple de sortie

   ```
   I0228 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
   I0228 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"
   I0228 09:36:40.769483       1 main.go:234] kmm/setup "msg"="starting manager"
   I0228 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"
   I0228 09:36:40.770025       1 internal.go:366] kmm "msg"="Starting server" "addr"={"IP":"::","Port":8081,"Zone":""} "kind"="health probe"
   I0228 09:36:40.770128       1 leaderelection.go:248] attempting to acquire leader lease openshift-kmm/kmm.sigs.x-k8s.io...
   I0228 09:36:40.784396       1 leaderelection.go:258] successfully acquired lease openshift-kmm/kmm.sigs.x-k8s.io
   I0228 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"
   ```

```
I0228 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"
I0228 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"
I0228 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"
I0228 09:36:40.785025       1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io" "controllerKind"="Module"
"source"="kind source: *v1.Node"
I0228 09:36:40.785039       1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="Module" "controllerGroup"="kmm.sigs.x-k8s.io" "controllerKind"="Module"
I0228 09:36:40.785458       1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="PodNodeModule" "controllerGroup"="" "controllerKind"="Pod" "source"="kind
source: *v1.Pod"
I0228 09:36:40.786947       1 controller.go:185] 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"="PreflightValidation" "source"="kind source: *v1.Job"
I0228 09:36:40.787488       1 controller.go:185] kmm "msg"="Starting EventSource"
"controller"="PreflightValidation" "controllerGroup"="kmm.sigs.x-k8s.io"
"controllerKind"="PreflightValidation" "source"="kind source: *v1beta1.Module"
I0228 09:36:40.787603       1 controller.go:185] kmm "msg"="Starting EventSource"
"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"
I0228 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:
*v1beta1.PreflightValidationOCP"
I0228 09:36:40.787853       1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="imagestream" "controllerGroup"="image.openshift.io"
"controllerKind"="ImageStream"
I0228 09:36:40.787879       1 controller.go:185] 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"
I0228 09:36:40.786489       1 controller.go:193] kmm "msg"="Starting Controller"
"controller"="PodNodeModule" "controllerGroup"="" "controllerKind"="Pod"
```

4.11.1.2. Collecte de données pour le KMM-Hub

Procédure

1. Rassembler les données pour le gestionnaire du concentrateur de l'opérateur KMM :

   a. Définir la variable **MUST_GATHER_IMAGE**:

   ```
   $ 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}')
   ```

   NOTE

   Utilisez le commutateur **-n <namespace>** pour spécifier un espace de noms si vous avez installé KMM dans un espace de noms personnalisé.

   b. Exécutez l'outil **must-gather**:

   ```
   $ oc adm must-gather --image="${MUST_GATHER_IMAGE}" -- /usr/bin/gather -u
   ```

2. Consulter les journaux de l'opérateur :

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

   Exemple 4.2. Exemple de sortie

   ```
   I0417 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
   I0417 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"
   I0417 11:34:12.376253       1 main.go:150] kmm-hub/setup "msg"="Adding controller" "name"="ManagedClusterModule"
   I0417 11:34:12.376621       1 main.go:186] kmm-hub/setup "msg"="starting manager"
   I0417 11:34:12.377690       1 leaderelection.go:248] attempting to acquire leader lease openshift-kmm-hub/kmm-hub.sigs.x-k8s.io...
   I0417 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"
   I0417 11:34:12.378222       1 internal.go:366] kmm-hub "msg"="Starting server" "addr"={"IP":"::","Port":8081,"Zone":""} "kind"="health probe"
   I0417 11:34:12.395703       1 leaderelection.go:258] successfully acquired lease openshift-kmm-hub/kmm-hub.sigs.x-k8s.io
   I0417 11:34:12.396334       1 controller.go:185] kmm-hub "msg"="Starting EventSource" "controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io" "controllerKind"="ManagedClusterModule" "source"="kind source: *v1beta1.ManagedClusterModule"
   I0417 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"
   ```

```
I0417 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"
I0417 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"
I0417 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"
I0417 11:34:12.396543       1 controller.go:193] kmm-hub "msg"="Starting Controller"
"controller"="ManagedClusterModule" "controllerGroup"="hub.kmm.sigs.x-k8s.io"
"controllerKind"="ManagedClusterModule"
I0417 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"
I0417 11:34:12.397221       1 controller.go:193] kmm-hub "msg"="Starting Controller"
"controller"="imagestream" "controllerGroup"="image.openshift.io"
"controllerKind"="ImageStream"
I0417 11:34:12.498335       1 filter.go:196] kmm-hub "msg"="Listing all
ManagedClusterModules" "managedcluster"="local-cluster"
I0417 11:34:12.498570       1 filter.go:205] kmm-hub "msg"="Listed
ManagedClusterModules" "count"=0 "managedcluster"="local-cluster"
I0417 11:34:12.498629       1 filter.go:238] kmm-hub "msg"="Adding reconciliation
requests" "count"=0 "managedcluster"="local-cluster"
I0417 11:34:12.498687       1 filter.go:196] kmm-hub "msg"="Listing all
ManagedClusterModules" "managedcluster"="sno1-0"
I0417 11:34:12.498750       1 filter.go:205] kmm-hub "msg"="Listed
ManagedClusterModules" "count"=0 "managedcluster"="sno1-0"
I0417 11:34:12.498801       1 filter.go:238] kmm-hub "msg"="Adding reconciliation
requests" "count"=0 "managedcluster"="sno1-0"
I0417 11:34:12.501947       1 controller.go:227] kmm-hub "msg"="Starting workers"
"controller"="imagestream" "controllerGroup"="image.openshift.io"
"controllerKind"="ImageStream" "worker count"=1
I0417 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
I0417 11:34:12.502285       1 imagestream_reconciler.go:50] kmm-hub "msg"="registered
imagestream info mapping" "ImageStream"={"name":"driver-
toolkit","namespace":"openshift"} "controller"="imagestream"
"controllerGroup"="image.openshift.io" "controllerKind"="ImageStream"
"dtkImage"="quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:df42b4785a7a662b30da53bdb0d206120cf4d24b45674227b16051ba4b7c393
4" "name"="driver-toolkit" "namespace"="openshift"
"osImageVersion"="412.86.202302211547-0" "reconcileID"="e709ff0a-5664-4007-8270-
49b5dff8bae9"
```