KubeCon | CloudNativeCon

North America 2023

KubeCon | CloudNativeCon
North America 2023

KMM: Your Swiss Army Knife for Kernel Modules on Kubernetes

Quentin Barrand (@qbarrand)
Hersh Pathak (@hershpa)

Agenda

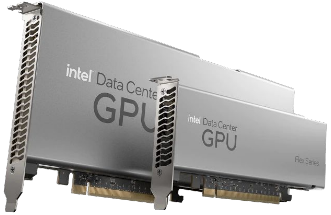
- Kernel Modules
- KMM Operator
- Use Case: Enabling Intel GPUs in Kubernetes
- Demo: Stable Diffusion text-to-image with KMM enabled Intel GPU
- Looking ahead: KMM 2.0
- Q&A

Why do I need kernel modules?

Kernel Modules?

Code written in C that extends the functionality of the kernel without the need to reboot the system

- Hardware drivers, virtual filesystems, additional system calls
- Built for a specific kernel version
- Can be loaded into and unloaded from the system
- Must be signed for Secure Boot systems



CSI

lustre®

Pain points

- Ideally, all kernel modules should be contributed upstream, but...
  - It is a long way to the kernel!
  - Latest hardware or A/B testing often require out-of-tree kmods
- Using kernel modules in production can be risky
  - Built for a specific ABI, need to rebuild when it changes
  - What if some symbol changes due to a critical CVE?
- How do we deploy modules on nodes?
  - Node customization is usually difficult to maintain

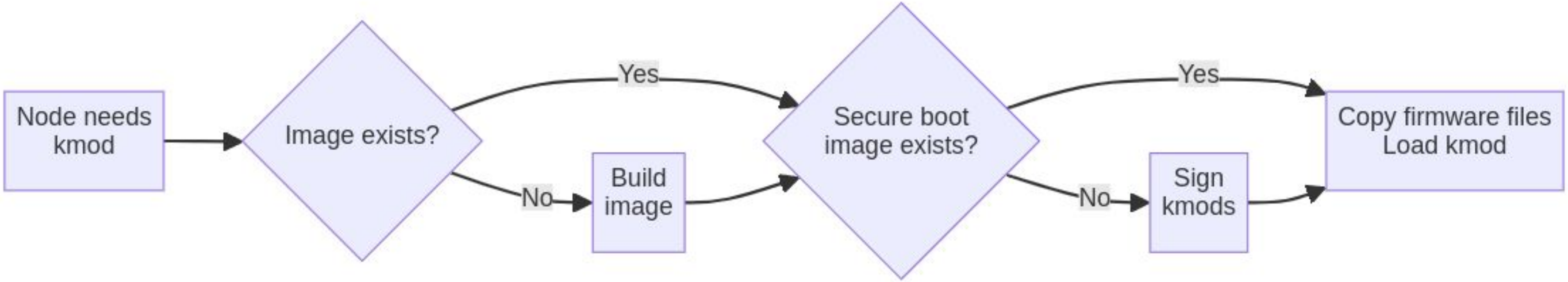
Enter KMM

Kernel Module Management operator

A standard consumption model for kernel modules on Kubernetes

- A SIG-Node project that builds, signs and loads kmods on your nodes
  - Monitors kernel versions in your cluster and loads modules as needed
  - Also runs your device plugin
- kmod images: a vehicle for kernel modules
  - Standard container images containing `.ko` files, firmwares and `modprobe`
  - One image can contain kmods for several kernel versions
- The `Module` custom resource maps kernel versions with kmod images
  - `literal` or `regexp`
  - One `Module` can accommodate many distros and kernel versions
  - You can use pre-built images for some kernels and build them for others!

KMM workflow



The simplest Module

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

      kernelMappings:
        - 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$'
          containerImage: 'some.other.registry/org/my-kmod:${KERNEL_FULL_VERSION}'

  selector:
    node-role.kubernetes.io/worker: ""
```

Building kmod images in-cluster

- Lazy builds, only once for all nodes
- Kaniko with user-provided Dockerfiles in ConfigMaps
- Build arguments and secrets are supported
  - Kernel version, module name and namespace provided by default

```
build:
  dockerfileConfigMap:
    name: my-kmod-dockerfile

  buildArgs: # Optional
    - name: ARG_NAME
      value: some-value

  secrets: # Optional
    # Mounted at /run/secrets/some-kubernetes-secret
    - name: some-kubernetes-secret

  baseImageRegistryTLS: # Optional
    insecure: false
    insecureSkipTLSVerify: false
```

Signing kmods in-cluster

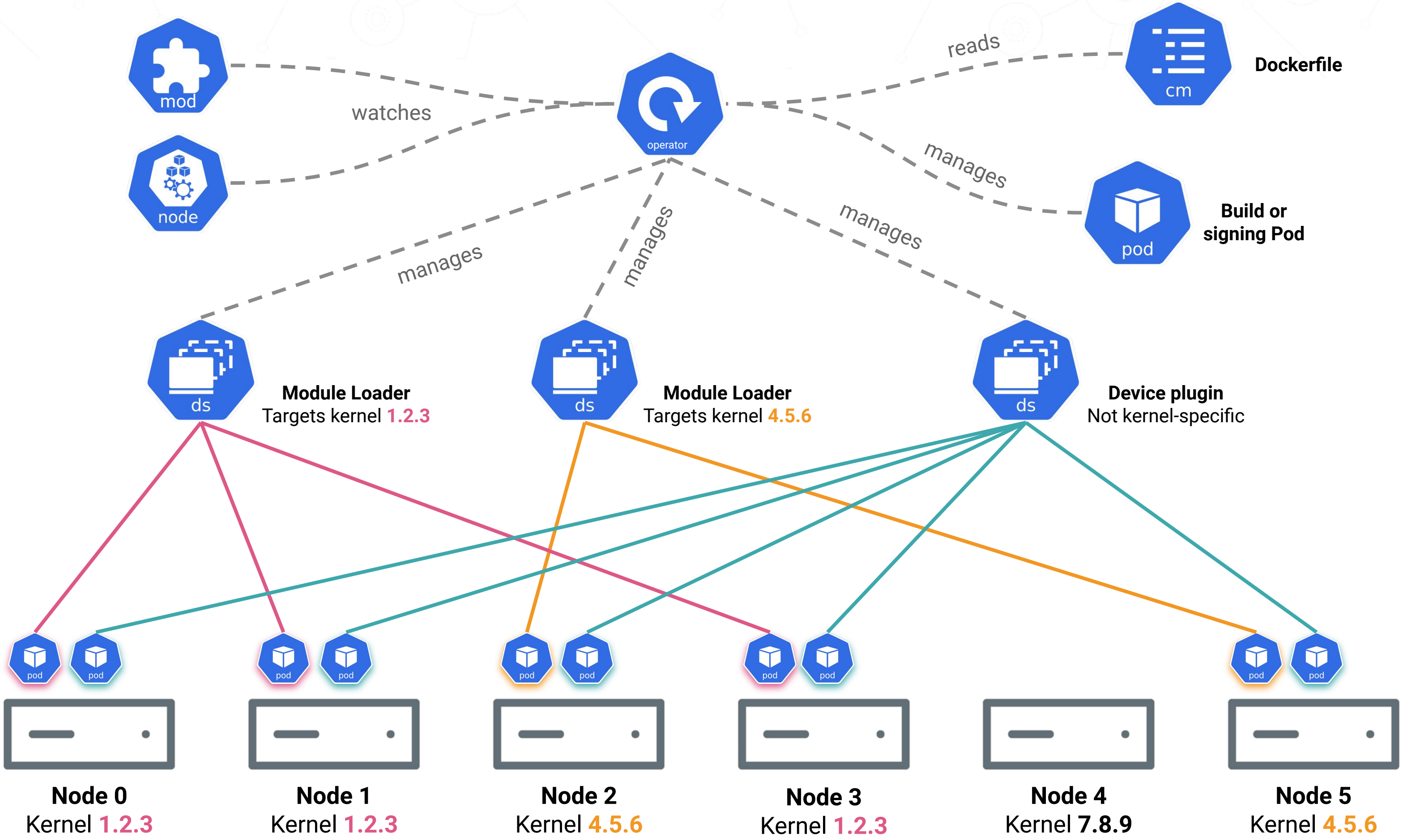
- A requirement for Secure Boot
- User-provided keys
- Implemented as a container image build with Kaniko

```yaml
sign:
  certSecret:
    name: cert-secret  # Required

  keySecret:
    name: key-secret  # Required

  filesToSign:
    - /opt/lib/modules/${KERNEL_FULL_VERSION}/my-kmod.ko

  registryTLS:
    insecure: false
    insecureSkipTLSVerify: false
```



Dockerfile

Build or signing Pod

Module Loader
Targets kernel 1.2.3

Module Loader
Targets kernel 4.5.6

Device plugin
Not kernel-specific

Node 0
Kernel 1.2.3

Node 1
Kernel 1.2.3

Node 2
Kernel 4.5.6

Node 3
Kernel 1.2.3

Node 4
Kernel 7.8.9

Node 5
Kernel 4.5.6

More features...

- Labeling nodes whenever modules are loaded
- Copies binary firmwares to the nodes
- Unloading any in-tree kmod before loading the out-of-tree one
- Soft dependencies (`softdep`)
- Preflight checks
- Custom kmod upgrade workflow

Use Case: Enabling Intel GPUs in Kubernetes

KMM + Intel GPU Use Case

- Use Case:
  - AI driven landscape demands latest HW enabling.
  - Latest drivers for optimal workload performance.
  - Chicken/Egg problem, delay between HW and SW readiness.
- **Problem:** No scalable, facilitated approach exists.
- **Production Environment:** Latest HW not truly “enabled” on Day 1.
  - Non-trivial HW/SW configuration
  - Complex node and kernel customization

KMM + Intel GPU Use Case

- **Why solve it today?** Leading edge XPU devices require enabling out-of-tree drivers to use XPUs
  - XPU Kernel drivers unavailable in OS distributions
  - Journey to upstream
  - Downstream lag
- **Goal:** Shift-left, HW and SW ready on Day 1
- **Impact:** KMM **accelerates** XPU enabling and **time to market**. Unlock optimized workloads and enable use case driven development.

Looking Ahead: KMM + Driver CI Pipeline

KMM powers a upcoming in-house Driver CI pipeline.

Pipeline addresses two scenarios:

1. New driver available
2. New kernel version

Future work: Facilitating seamless driver upgrades.

Value: Customers can enable GPUs in seconds with pre-built images.

1. Deploy pre-built image with KMM
2. GPU device plugin extends GPU resource for workload consumption.

Demo: Enabling Intel GPUs with KMM on Kubernetes

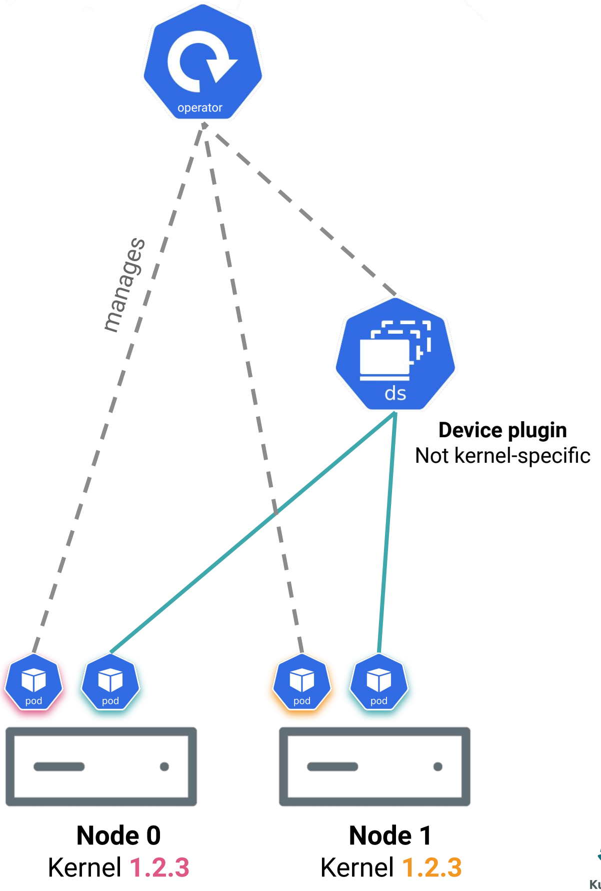
Demo: Enabling Intel GPUs with KMM

- Single node minikube cluster
    - 4th Gen Intel Xeon + Intel GPU Flex 170 on DevCloud
    - Ubuntu® 22.04 Server (5.15 generic)
- One Jupyter Pod
    - OpenVINO
    - Runtime libraries
    - 20 cores, 64G RAM
- Build and load the driver with KMM
- Run a text-to-image notebook w/ Stable Diffusion with OpenVino on CPU & GPU

Demo code on GitHub:

Looking ahead: KMM 2.0

- Long-running DaemonSets
  ➡️ short-lived Worker Pods
  - Pull kmod image, run `modprobe`, exit
  - Better reliability
  - Lower footprint
- Improved binary firmware support
  - Worker Pod sets the lookup path via sysfs
- Coming later this month!



Wrapping up

- KMM is a Kubernetes operator that loads kernel modules on nodes
  - Can also build kmods and sign them for Secure Boot
- A standard consumption model
  - Flexible API that addresses multiple kernels and distros at once
  - Labels nodes whenever modules are loaded
- v1.1 available now
  - 2.0 coming in November with a much smaller footprint; stay tuned!
- Visit the Intel booth for more demos!

Questions? Comments? Suggestions?
We love feedback and contributions!

Join us at:

#sig-node-kmm on Slack

kmm.sigs.k8s.io

github.com/kubernetes-sigs/kernel-module-management

github.com/intel/intel-technology-enabling-for-openshift

intel.github.io/intel-technology-enabling-for-openshift/

**Please scan the QR Code above
to leave feedback on this session**