

AI 进阶指南：GPU 故障不求人

主讲人：朱争光

Content

目录

1. 故障排查思路

2. 安装使用

3. GPU Operator 原理

4. 故障排查指南&常见故障

Part 01

故障排查思路

■ 如何维护好一个系统？

个人总结的知识学习方式

WHAT

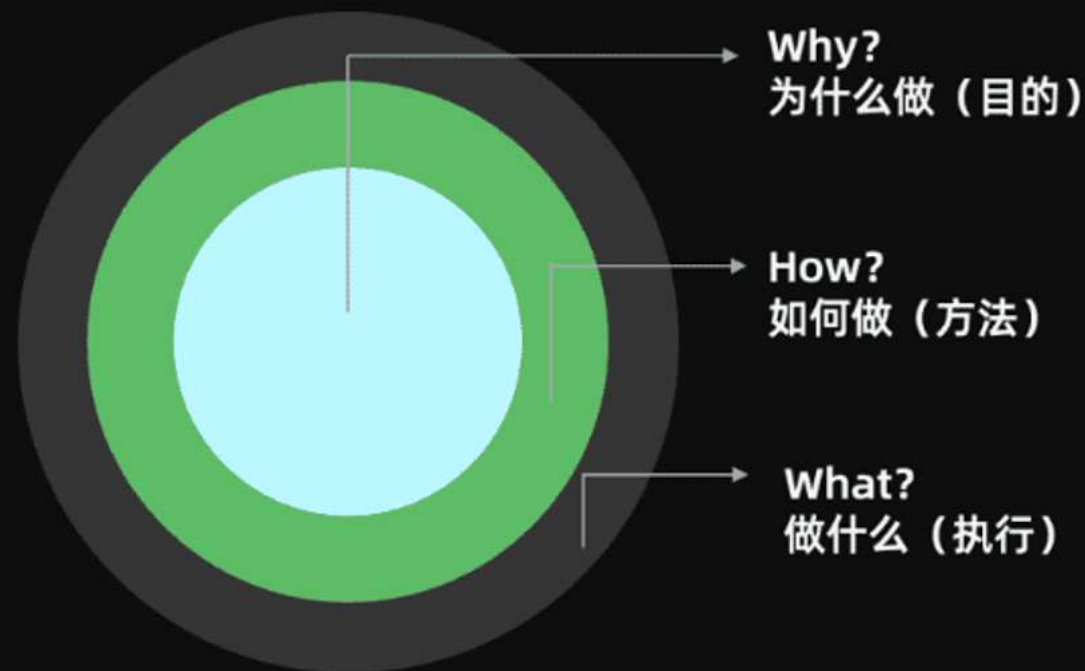
知道是什么 --- 了解

HOW

怎么（安装）使用 --- 熟悉

WHY

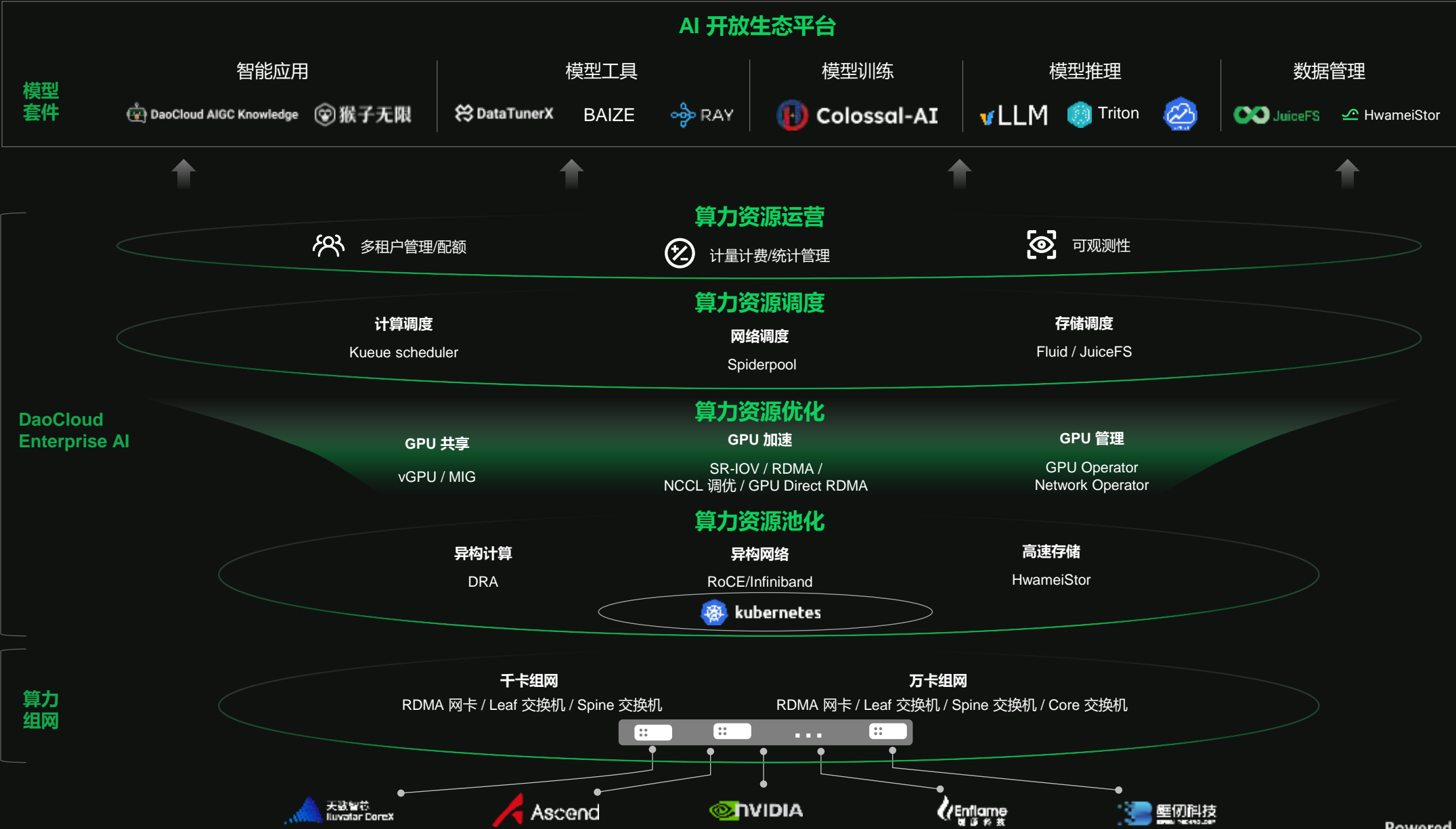
底层是如何实现的？ --- 精通



3W黄金圈法则

AI 应用通过容器**高效使用 GPU** 需要啥？

d.run 产品全景图

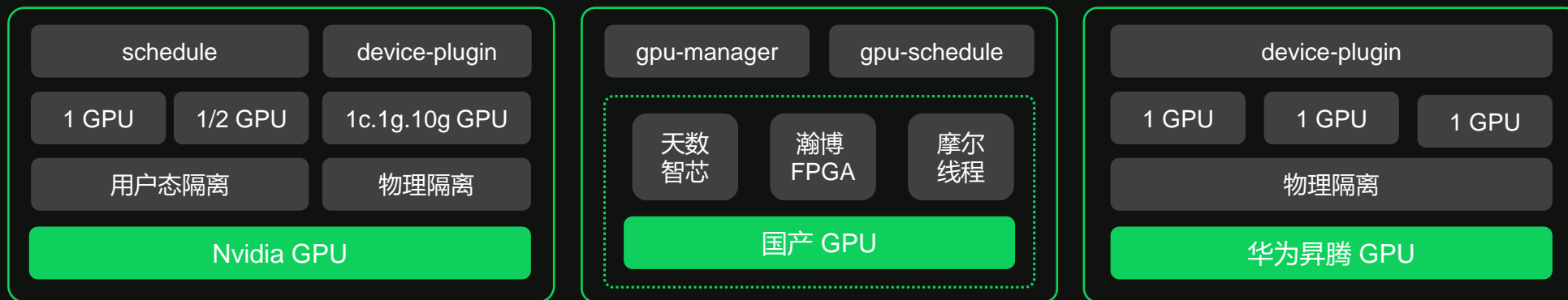


■ 算力资源池化现状

资源调度层



资源隔离层

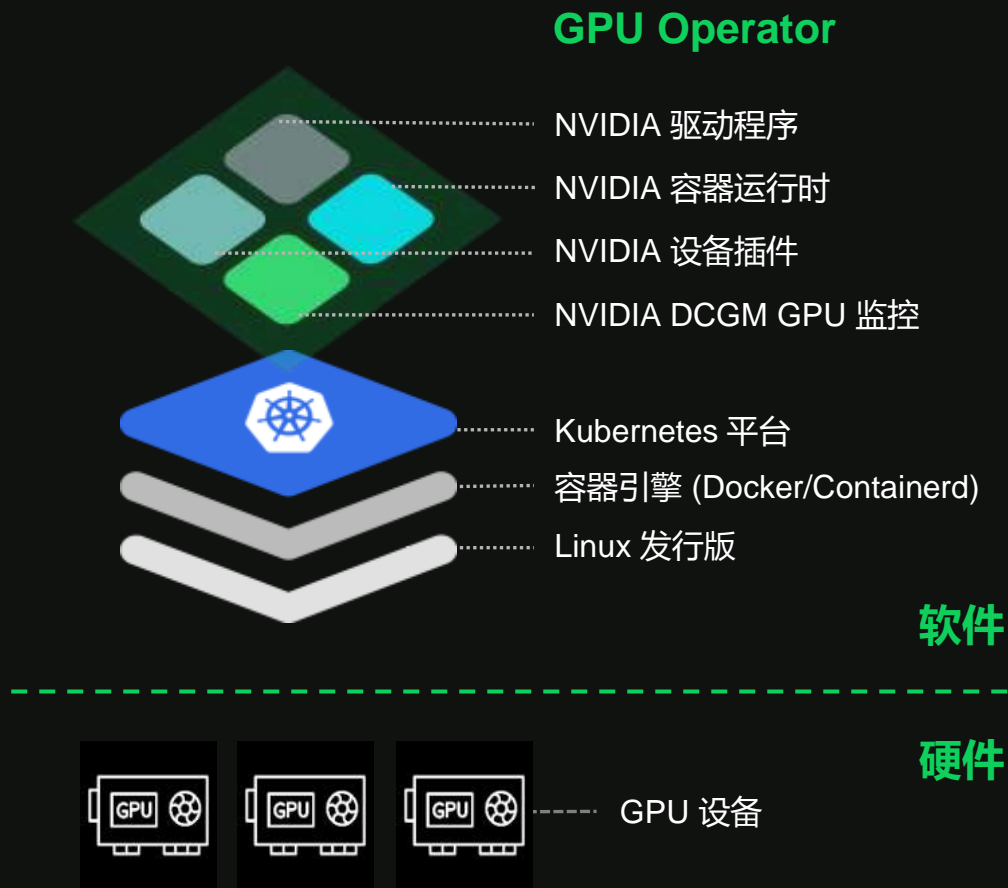


GPU 池化各家做法大致类似，今天我们以目前应用最广泛的 Nvidia Operator 为例来讲解

■ 这次只聊 NVIDIA GPU Operator 🤗

GPU Operator - 自动管理并提供 GPU 所需的所有 NVIDIA 软件组件

- **NVIDIA 设备插件** - 通过设备插件机制将 GPU 公开给 Kubelet
- **NVIDIA 容器工具包** : 实现容器化环境中与 GPU 进行交互
- **GPU 驱动程序** : Nvidia 驱动程序组件允许从容器进行驱动安装
- **NVIDIA GPU 功能发现** : 检测并标记启用 GPU 的节点
- **NVIDIA DCGM GPU 监控** : 采集 GPU 指标



Part 02

安装使用

■ 安装GPU Operator

1. Add the NVIDIA Helm repository:

```
$ helm repo add nvidia https://helm.ngc.nvidia.com/nvidia \
  && helm repo update
```

2. Install the GPU Operator.

- Install the Operator with the default configuration:

```
$ helm install --wait --generate-name \
  -n gpu-operator --create-namespace \
  nvidia/gpu-operator
```

推荐：

系统：ubuntu 22.04

容器：containerD

K8s: 1.29+

driver.enabled
driver.usePrecompiled
toolkit.enabled
driver.version
toolkit.version

官网: <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/getting-started.html>

■ 安装完成后的效果

The screenshot displays the DaoCloud management interface for a cluster named 'workload-002' in the 'gpu-operator' namespace. The left sidebar shows the navigation menu with '工作负载' (Workloads) selected. The main content area is divided into two panels. The left panel shows a list of workloads under '无状态负载' (Stateless Workloads), including 'gpu-operator', 'gpu-operator-node-feature-discovery-gc', and 'gpu-operator-node-feature-discovery-master'. The right panel shows the '守护进程' (Guard Processes) section, which contains a table of services. The 'nvidia-operator-validator' service is highlighted with a red box and a green callout box, indicating it is running.

集群: workload-002 / 命名空间: gpu-operator / 守护进程

守护进程

输入工作负载名称搜索

工作负载名称	工作负载别名	状态
<input type="checkbox"/> nvidia-driver-daemonset-5.15.0-94-generic-ubuntu22.04	-	运行中
<input type="checkbox"/> gpu-feature-discovery	-	运行中
<input type="checkbox"/> nvidia-container-toolkit-daemonset	-	运行中
<input type="checkbox"/> nvidia-dcgm-exporter	-	运行中
<input type="checkbox"/> nvidia-device-plugin-daemonset	-	运行中
<input type="checkbox"/> nvidia-mig-manager	-	运行中
<input type="checkbox"/> nvidia-operator-validator	-	运行中
<input type="checkbox"/> gpu-operator-node-feature-discovery-worker	-	运行中

共 8 项

■ 如何使用

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vectoradd
spec:
  restartPolicy: OnFailure
  containers:
  - name: cuda-vectoradd
    image: "nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubuntu2
    resources:
      limits:
        nvidia.com/gpu: 1
```

nvidia.com/gpu: 1

■ 测试

nvidia-smi

workload-002

集群: workload-002 / 命名空间: default / 无状态负载 / gpu-demo

控制台 监控 日志 ...

基本信息

gpu-demo-6589bc7dbc-tfh7n-container-1

```
root@gpu-demo-6589bc7dbc-tfh7n:/# nvidia-smi
Tue Sep 24 10:35:28 2024
```

NVIDIA-SMI 535.154.05			Driver Version: 535.154.05		CUDA Version: 12.3		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute M.
							MIG M.
0	Tesla P4		On	00000000:03:01.0	Off		0
N/A	40C	P8	6W / 75W	0MiB / 7680MiB		0%	Default
							N/A

```
Processes:
```

GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
	ID	ID				
No running processes found						

```
root@gpu-demo-6589bc7dbc-tfh7n:/#
```

Part 03

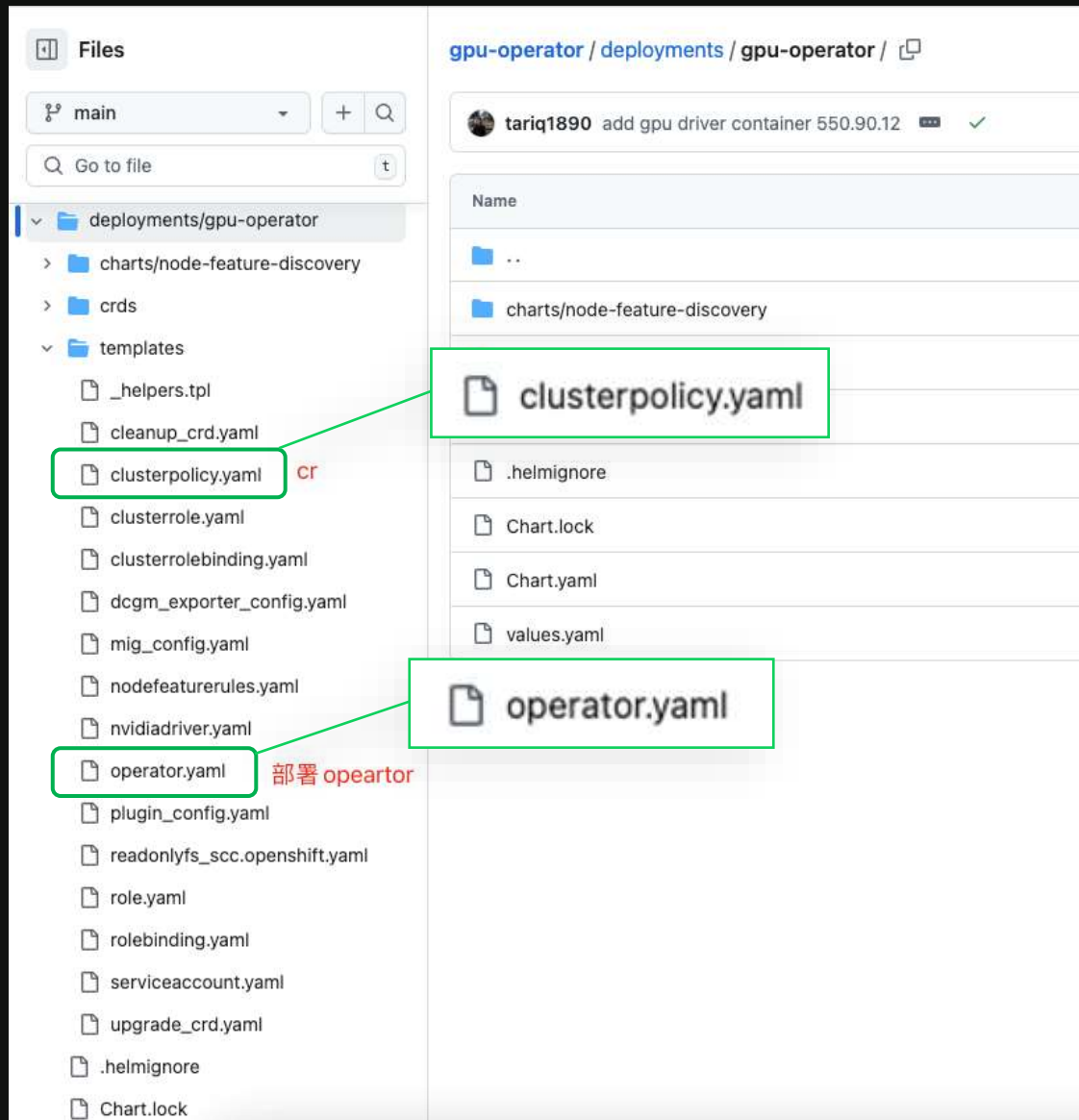
GPU Operator 原理

■ 安装命令底层层执行了啥？

Helm 包大致逻辑：

1. 安装 operator 和 node-feature-discovery 模块
2. 创建 cr 和配置文件

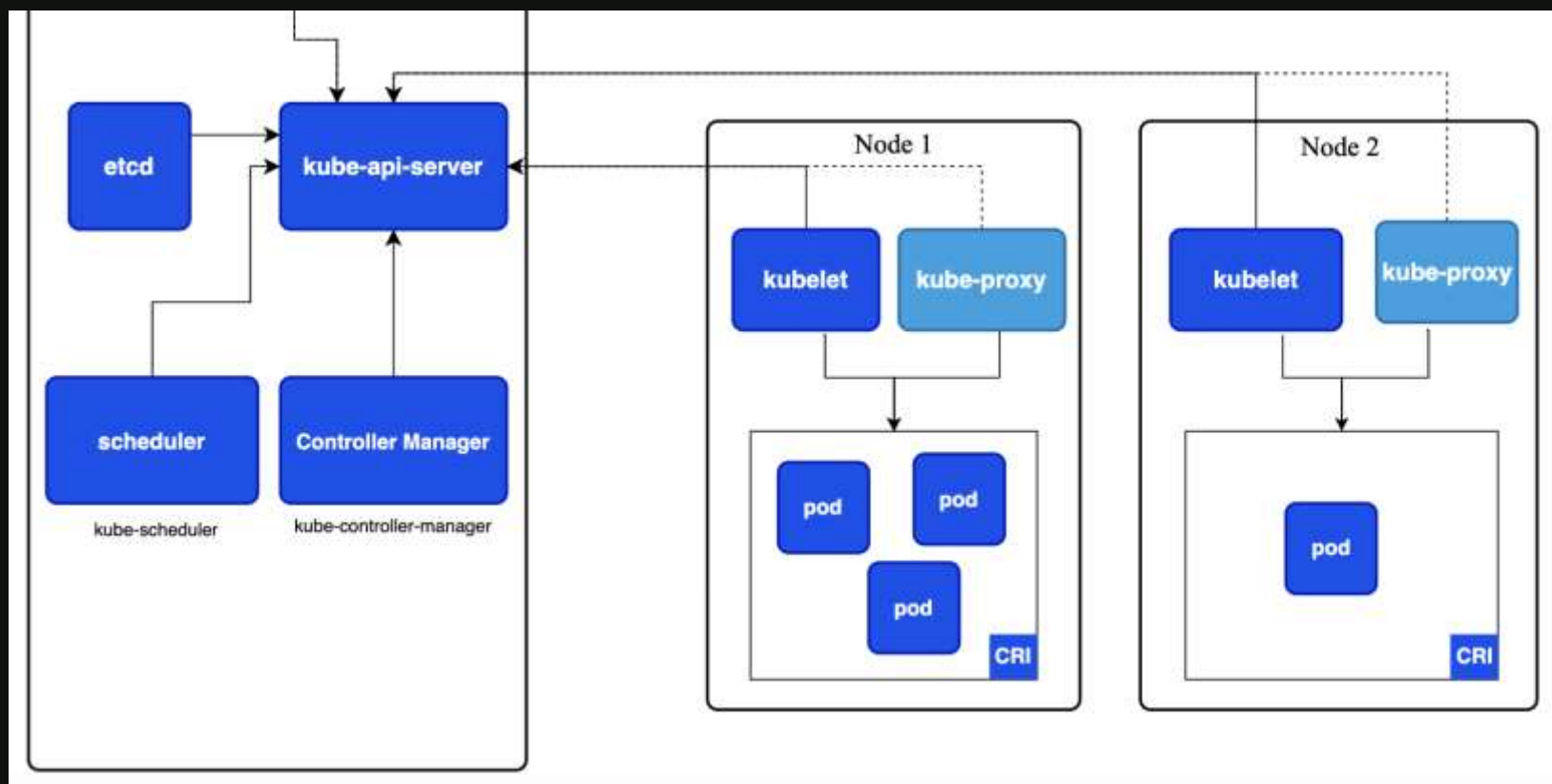
然后 operator 会根据 cr 完成剩余安装



■ GPU Operator 如何完成剩余任务的？

Gpu operator: 根据 ClusterPolicy 的配置，创建各组件的 ds，一般是：

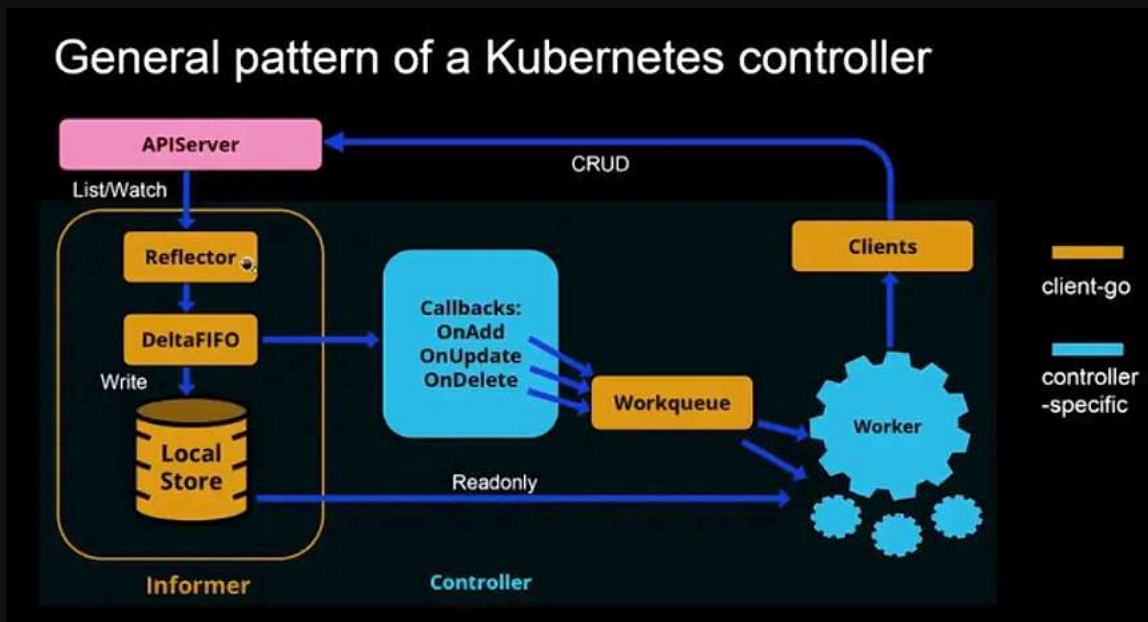
- Driver
- Container-toolkit
- Device-plugin
- Node-feature discovery
- Dcgm-exporter
- Validator



■ GPU Operator 如何完成剩余任务的？

Gpu operator: 根据 ClusterPolicy 的配置, 创建各组件的 ds, 一般是:

- Driver
- Container-toolkit
- Device-plugin
- Node-feature discovery
- Dcgm-exporter
- Validator



```
root@m-10-17-16-18:~# kubectl get clusterpolicies.nvidia.com/cluster-policy
apiVersion: nvidia.com/v1
kind: ClusterPolicy
metadata:
  annotations:
    meta.helm.sh/release-name: gpu-operator
    meta.helm.sh/release-namespace: gpu-operator
  creationTimestamp: "2024-08-30T05:03:24Z"
  generation: 3
  labels:
    app.kubernetes.io/component: gpu-operator
    app.kubernetes.io/instance: gpu-operator
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: gpu-operator
    app.kubernetes.io/version: v23.9.0
    helm.sh/chart: gpu-operator-v23.9.0
  name: cluster-policy
  resourceVersion: "12931634"
  uid: 68be3737-14d5-43f1-9041-63c4b58606c6
spec:
  ccManager:
    defaultMode: "off"
    enabled: false
    env: []
    image: nvidia/cloud-native/k8s-cc-manager
    imagePullPolicy: IfNotPresent
    repository: 10.17.16.10/nvcr.m.daocloud.io
    version: v0.1.1
  cdi:
    default: true
    enabled: true
  daemonsets:
    labels:
      app.kubernetes.io/managed-by: gpu-operator
      helm.sh/chart: gpu-operator-v23.9.0
    priorityClassName: system-node-critical
    rollingUpdate:
      maxUnavailable: "1"
    tolerations:
      - effect: NoSchedule
        key: nvidia.com/gpu
        operator: Exists
    updateStrategy: RollingUpdate
  dcgm:
    enabled: false
    hostPort: 5555
    image: nvidia/cloud-native/dcgm
    imagePullPolicy: IfNotPresent
    repository: 10.17.16.10/nvcr.m.daocloud.io
    version: 3.2.6-1-ubuntu20.04
```

■ Driver 是如何装起来的？

目前 driver 有两种安装方式

1. 通过 clusterPolicy 设置：一般场景
2. 通过 NVIDIADriver 设置：针对不同节点 gpu 和系统存在差异等场景

容器做的事情

1. 初始化: 卸载驱动

driver-manager uninstall_driver

2. 启动: 安装驱动

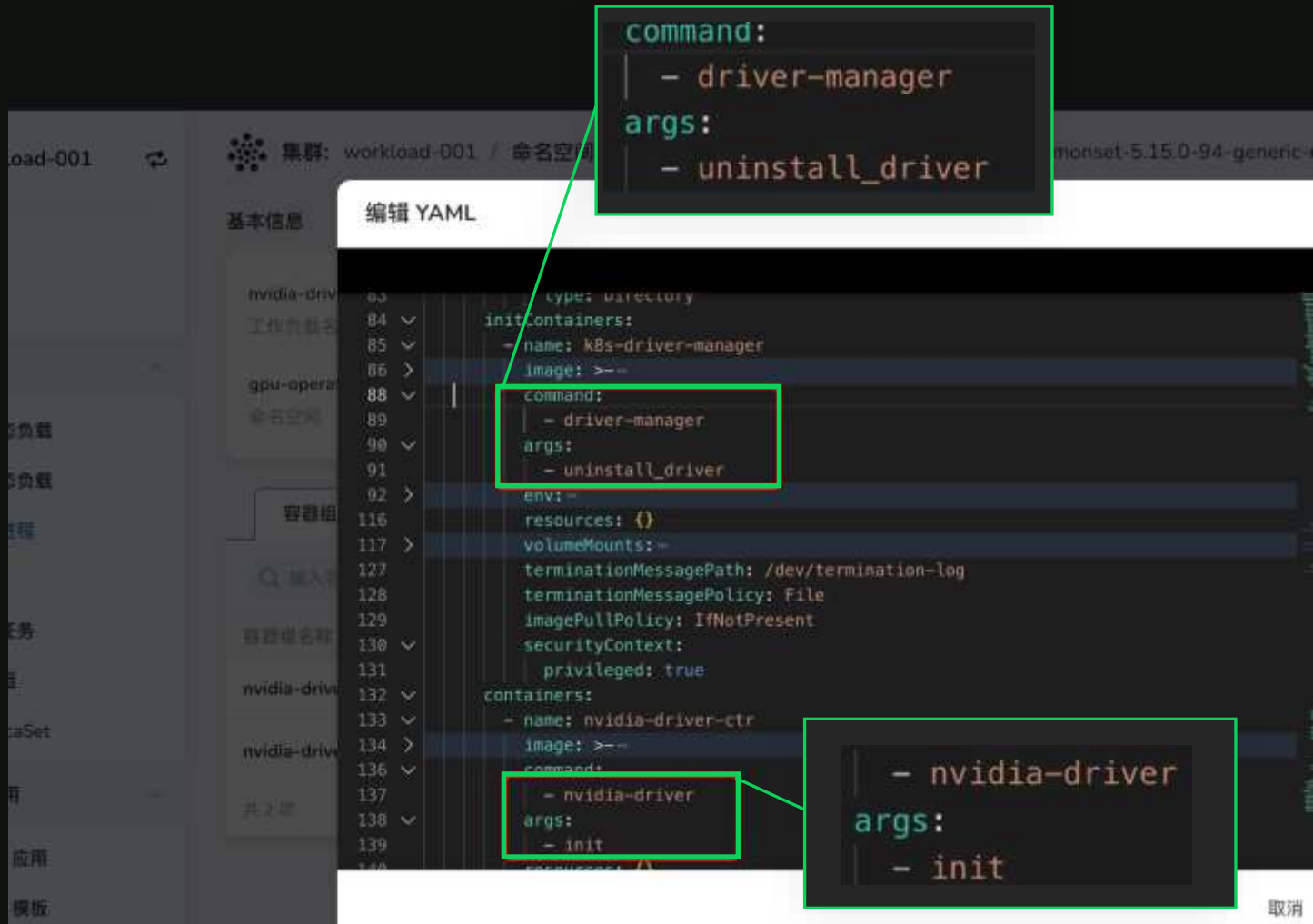
nvidia-driver init

3. 成功后: 创建

/run/nvidia/validations/.driver-ctr-ready

4. 删除 pod 前：

删除/run/nvidia/validations/.driver-ctr-ready



■ Driver 预编译安装逻辑-离线安装

容器启动后, 执行 [nvidia-driver](#) init

安装逻辑：

1. 卸载内核模块：nvidia-drm, nvidia-modeset, nvidia-uvm, nvidia-peermem, nvidia
2. umount /run/nvidia/driver
3. _install_driver: 根据OPEN_KERNEL_MODULES_ENABLED变量安装开源或闭源驱动
(因为镜像构建时已经执行过, 包都缓存了, 所以这里不需要联网)
4. _load_driver: 启用内核模块: modprobe nvidia ...;启动nvidia-persistenced, nv-fabricmanager

5. mount /sys 和 mount / /run/nvidia/driver

<https://docs.nvidia.com/datacenter/cloud-native/openshift/latest/gpu-operator-with-precompiled-drivers.html>

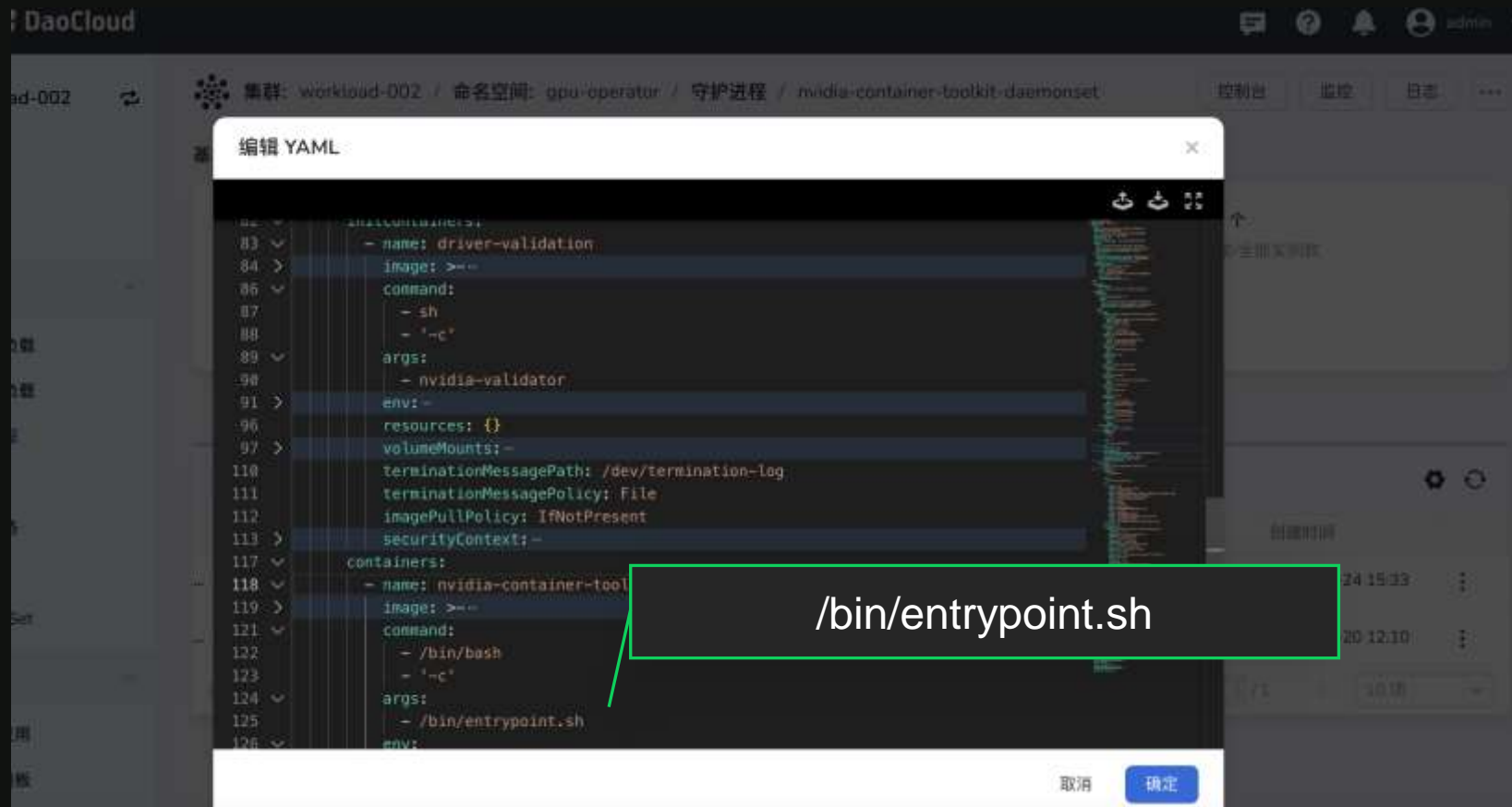
<https://github.com/NVIDIA/gpu-driver-container/blob/main/ubuntu22.04/precompiled/nvidia-driver>

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/driver/tags>

■ Container toolkit 安装解释

/work/toolkit 执行内容:

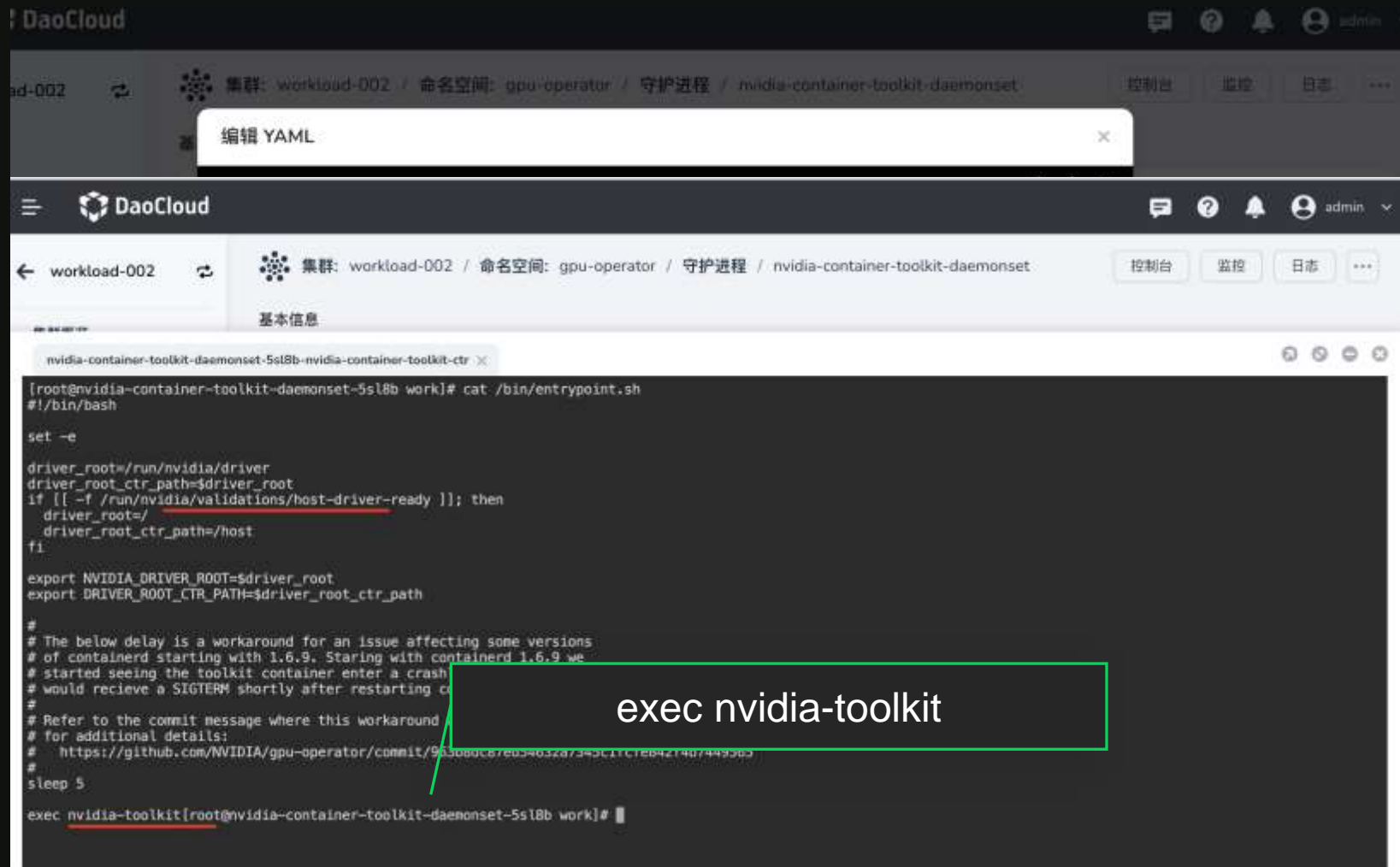
1. 安装 runtime
2. 修改 containerd 的配置
增加 runtime 并且设置为默认
3. 触发 containerd 重启



■ Container toolkit 安装解释

/work/toolkit 执行内容:

1. 安装 runtime
2. 修改 containerd 的配置
增加 runtime 并且设置为默认
3. 触发 containerd 重启



```
DaoCloud

集群: workload-002 / 命名空间: gpu-operator / 守护进程: nvidia-container-toolkit-daemonset

编辑 YAML

workload-002 集群: workload-002 / 命名空间: gpu-operator / 守护进程: nvidia-container-toolkit-daemonset 控制台 监控 日志

基本信息

nvidia-container-toolkit-daemonset-5sl8b-nvidia-container-toolkit-ctr X

[root@nvidia-container-toolkit-daemonset-5sl8b work]# cat /bin/entrypoint.sh
#!/bin/bash

set -e

driver_root=/run/nvidia/driver
driver_root_ctr_path=$driver_root
if [[ -f /run/nvidia/validations/host-driver-ready ]]; then
    driver_root=/
    driver_root_ctr_path=/host
fi

export NVIDIA_DRIVER_ROOT=$driver_root
export DRIVER_ROOT_CTR_PATH=$driver_root_ctr_path

#
# The below delay is a workaround for an issue affecting some versions
# of containerd starting with 1.6.9. Starting with containerd 1.6.9 we
# started seeing the toolkit container enter a crash
# would receive a SIGTERM shortly after restarting c
#
# Refer to the commit message where this workaround
# for additional details:
# https://github.com/NVIDIA/gpu-operator/commit/963080c67e054032a7345c11c1eb4214074495b3
#
sleep 5

exec nvidia-toolkit [root@nvidia-container-toolkit-daemonset-5sl8b work]#
```


■ Container toolkit 安装解释

/work/toolkit 执行内容:

1. 安装 runtime
2. 修改 containerd 的配置
增加 runtime 并且设置为默认
3. 触发 containerd 重启

```
/etc/containerd/config.toml
root@m-10-17-16-18:~# cat /etc/containerd/config.toml
oom_score = 0
root = "/var/lib/containerd"
state = "/run/containerd"
version = 2
```

```
[debug]
  level = "info"
```

```
[grpc]
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
```

```
[metrics]
  address = ""
  grpc_histogram = false
```

```
[plugins]
```

```
[plugins."io.containerd.grpc.v1.cri"]
  enable_unprivileged_icmp = false
  enable_unprivileged_ports = false
  max_container_log_line_size = -1
  sandbox_image = "10.17.16.10/regist
```

root@m-10-17-16

```
default_runtime_name = "nvidia"
```

```
[plugins."io.containerd.grpc.v1.cri".containerd]
  default_runtime_name = "nvidia"
  discard_unpacked_layers = true
  snapshotter = "overlayfs"
```

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]
```

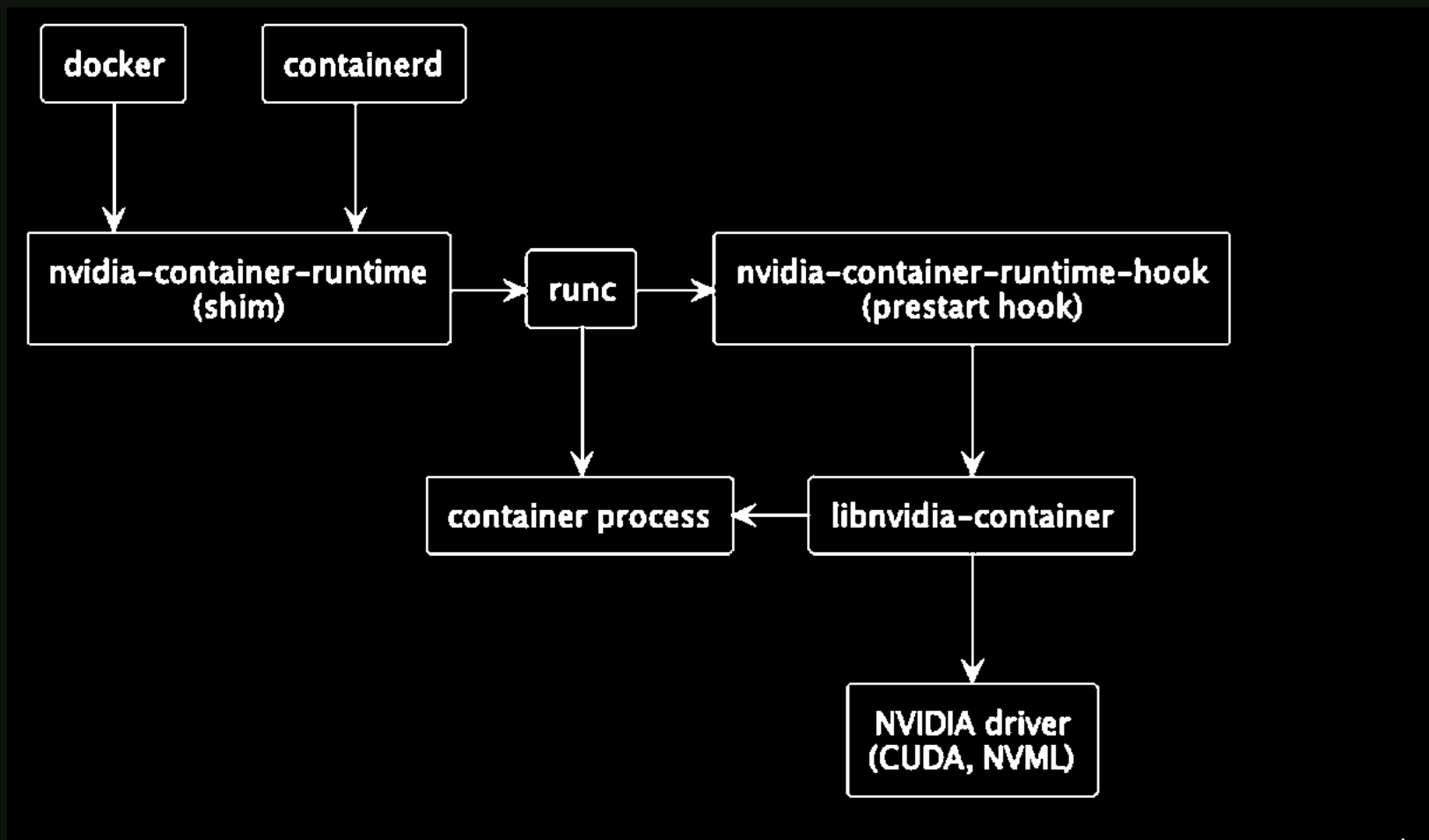
```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]
  base_runtime_spec = "/etc/containerd/cri-base.json"
  container_annotations = ["nvidia.cdi.k8s.io/*"]
  runtime_engine = ""
  runtime_root = ""
  runtime_type = "io.containerd.runc.v2"
```

```
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]
  BinaryName = "/usr/local/nvidia/toolkit/nvidia-container-runtime"
```

■ Container toolkit : 容器如何能运行GPU的？

作用：

让容器内能够使用 gpu



<https://github.com/NVIDIA/nvidia-container-toolkit/blob/main/tools/container/README.md>

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/arch-overview.html>

Hook 概念?

■ container toolkit 测试命令

```
export PATH=$PATH:/usr/local/nvidia/toolkit
```

```
echo "/run/nvidia/driver/usr/lib/x86_64-linux-gnu" > /etc/ld.so.conf.d/nvidia.conf
```

```
ldconfig
```

```
nerdctl run -it --rm --runtime=/usr/local/nvidia/toolkit/nvidia-container-runtime \
```

```
--gpus all m.daocloud.io/docker.io/nvidia/cuda:12.3.1-base-ubuntu20.04 nvidia-smi
```

```
root@m-10-17-16-18:~# nerdctl run -it --rm --runtime=/usr/local/nvidia/toolkit/nvidia-container-runtime --gpus all m.daocloud.io/docker.io/nvidia/cuda:12.3.1-base-ubuntu20.04 bash
root@f15313c40a62:/# df -h

```

Filesystem	Size	Used	Avail	Use%	Mounted on
overlay	97G	23G	75G	23%	/
tmpfs	64M	0	64M	0%	/dev
shm	64M	0	64M	0%	/dev/shm
/dev/mapper/vg0-lv--0	97G	23G	75G	23%	/etc/hosts
tmpfs	7.9G	12K	7.9G	1%	/proc/driver/nvidia
overlay	97G	23G	75G	23%	/usr/bin/nvidia-smi
tmpfs	64M	0	64M	0%	/dev/nvidia0
tmpfs	7.9G	0	7.9G	0%	/proc/acpi
tmpfs	7.9G	0	7.9G	0%	/sys/firmware
tmpfs	7.9G	0	7.9G	0%	/sys/devices/virtual/powercap
tmpfs	7.9G	0	7.9G	0%	/proc/scsi

```
root@f15313c40a62:/# exit
exit
root@m-10-17-16-18:~# nerdctl run -it --rm m.daocloud.io/docker.io/nvidia/cuda:12.3.1-base-ubuntu20.04 bash
root@5928cc845a92:/# df -h

```

Filesystem	Size	Used	Avail	Use%	Mounted on
overlay	97G	23G	75G	23%	/
tmpfs	64M	0	64M	0%	/dev
shm	64M	0	64M	0%	/dev/shm
/dev/mapper/vg0-lv--0	97G	23G	75G	23%	/etc/hosts
tmpfs	7.9G	0	7.9G	0%	/proc/acpi
tmpfs	7.9G	0	7.9G	0%	/sys/firmware
tmpfs	7.9G	0	7.9G	0%	/sys/devices/virtual/powercap
tmpfs	7.9G	0	7.9G	0%	/proc/scsi

```
root@5928cc845a92:/#
```

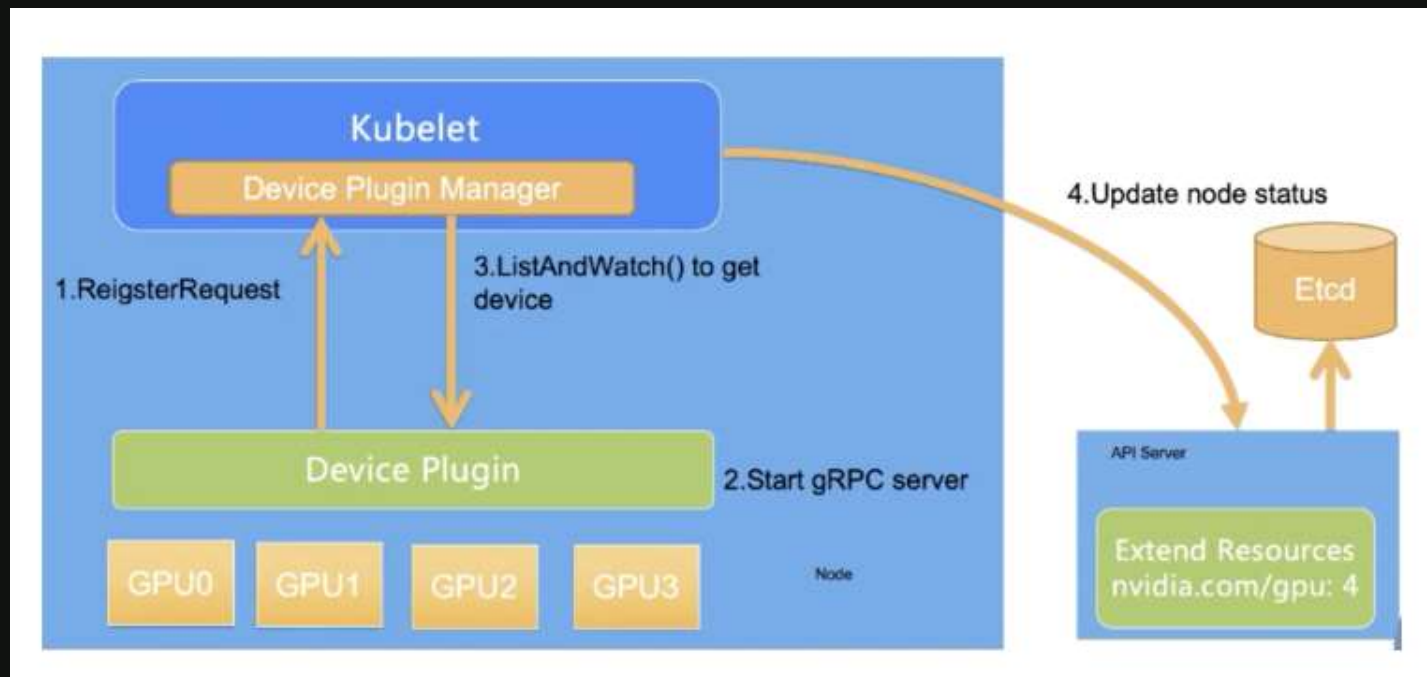
/proc/driver/nvidia
/usr/bin/nvidia-smi
/dev/nvidia0

■ K8s device plugin: K8s 如何运行GPU的

K8S 提供了device plugin 机制，可以扩展自定义的设备

大概的过程：

1. 插件程序按实现规范启动 grpc 服务
2. 然后插件程序向 kubelet 注册自己
3. kubelet 调用插件获取设备信息，更新到 node
4. 当部署 pod 指定了某个设备时，kubelet 调用插件程序完成容器附加设备的动作



■ K8s device plugin 检查方式

kubectl get node m-10-17-16-18 -oyaml|grep allocatable: -C 10

可以看到对应的设备信息

```
root@m-10-17-16-18:~# kubectl get node m-10-17-16-18 -oyaml|grep allocatable: -C 10
  name: m-10-17-16-18
  resourceVersion: "14012050"
  uid: 3d8b2ffd-8479-43b9-a3e4-8fa455a8ecdc
spec: {}
status:
  addresses:
  - address: 10.17.16.18
    type: InternalIP
  - address: m-10-17-16-18
    type: Hostname
  allocatable:
    cpu: 7800m
    ephemeral-storage: "93639068313"
    hugepages-1Gi: "0"
    hugepages-2Mi: "0"
    memory: 15745052Ki
    nvidia.com/gpu: "1"
    pods: "110"
  capacity:
    cpu: "8"
```

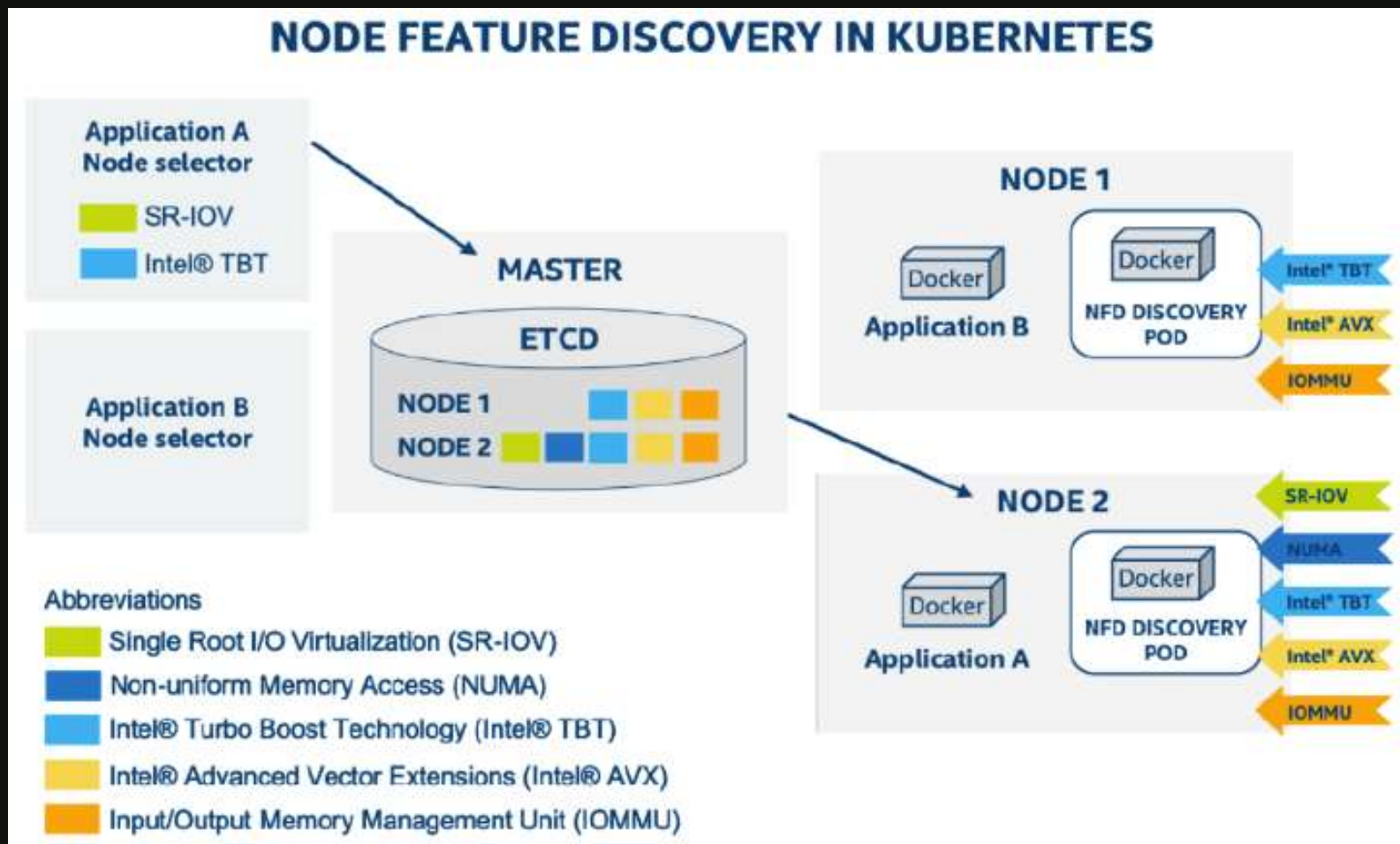

nvidia.com/gpu: "1"

■ Feature discovery : K8s 如何选择特定的GPU卡

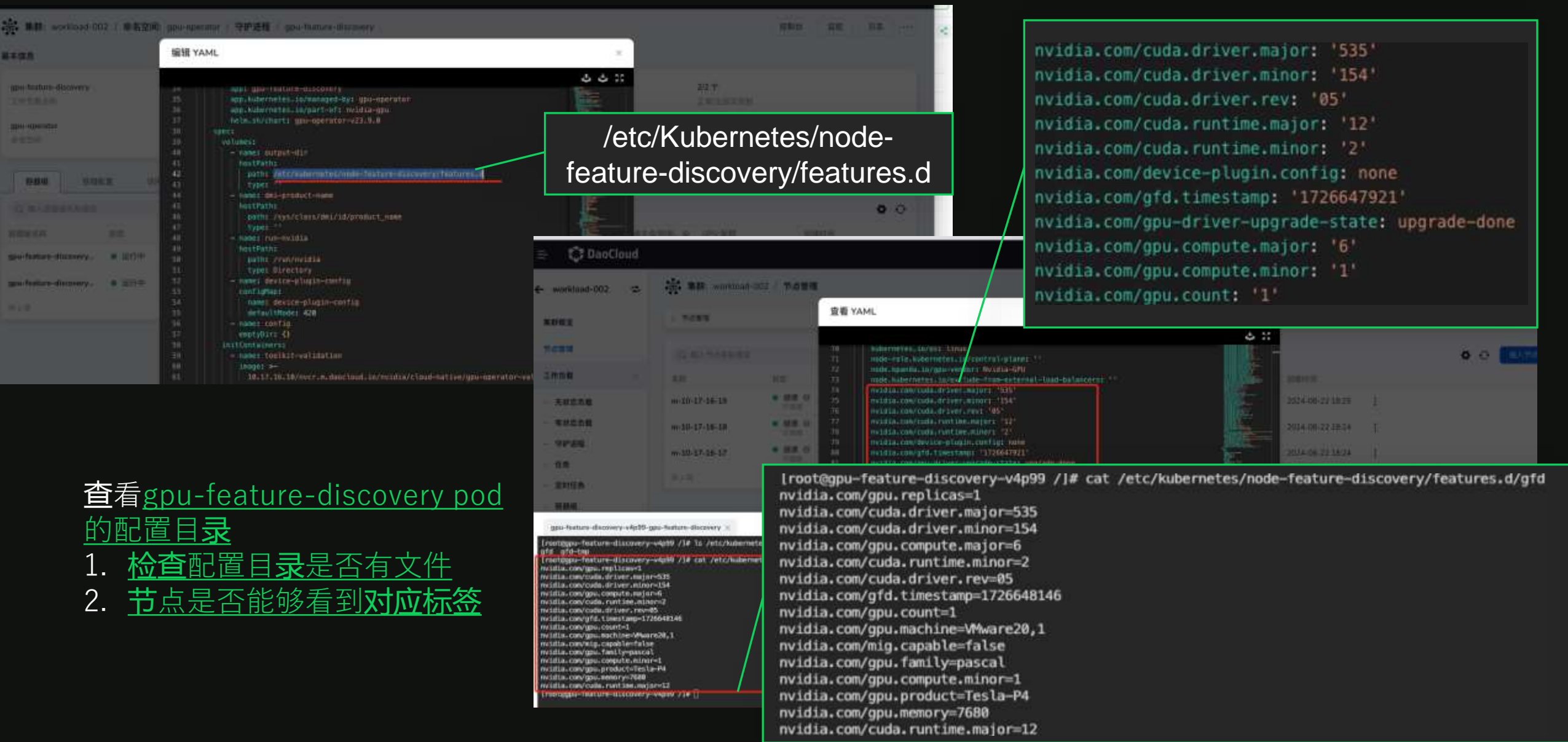
NFD-Master : 是一个负责与 kubernetes API Server 通信的 Deployment Pod, 它从 NFD-Worker 接收节点特性并相应地修改 Node 资源对象 (标签、注解)。

NFD-Worker : 是一个负责对 Node 的特性能力进行检测的 Daemon Pod, 然后它将信息传递给 NFD-Master, NFD-Worker 应该在每个 Node 上运行。

NFD 插件: 根据规范, 把自己的设备特性写入到特定目录



■ Feature discovery 安装后如何确认和排查



编辑 YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-feature-discovery
  namespace: kube-system
spec:
  volumes:
  - name: output-dir
    hostPath:
      path: /etc/Kubernetes/node-feature-discovery/features.d
      type: DirectoryOrCreate
  - name: dev-dir
    hostPath:
      path: /dev/nvidia
      type: DirectoryOrCreate
  - name: device-plugin-config
    configMap:
      name: device-plugin-config
      defaultMode: 420
  - name: conf-dir
    emptyDir: {}
  initContainers:
  - name: tonkit-validation
    image: 7c-10.17.16.16/ncr.a.daocloud.io/ncr/a/cloud-native/gpu-operator-v23.9.0
```

/etc/Kubernetes/node-feature-discovery/features.d

```
nvidia.com/cuda.driver.major: '535'
nvidia.com/cuda.driver.minor: '154'
nvidia.com/cuda.driver.rev: '05'
nvidia.com/cuda.runtime.major: '12'
nvidia.com/cuda.runtime.minor: '2'
nvidia.com/device-plugin.config: none
nvidia.com/gfd.timestamp: '1726647921'
nvidia.com/gpu-driver-upgrade-state: upgrade-done
nvidia.com/gpu.compute.major: '6'
nvidia.com/gpu.compute.minor: '1'
nvidia.com/gpu.count: '1'
```

查看 YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-feature-discovery-v4p99
  namespace: kube-system
spec:
  volumes:
  - name: output-dir
    hostPath:
      path: /etc/Kubernetes/node-feature-discovery/features.d
      type: DirectoryOrCreate
  - name: dev-dir
    hostPath:
      path: /dev/nvidia
      type: DirectoryOrCreate
  - name: device-plugin-config
    configMap:
      name: device-plugin-config
      defaultMode: 420
  - name: conf-dir
    emptyDir: {}
  initContainers:
  - name: tonkit-validation
    image: 7c-10.17.16.16/ncr.a.daocloud.io/ncr/a/cloud-native/gpu-operator-v23.9.0
```

查看

```
root@gpu-feature-discovery-v4p99 /# cat /etc/kubernetes/node-feature-discovery/features.d/gfd
nvidia.com/gpu.replicas=1
nvidia.com/cuda.driver.major=535
nvidia.com/cuda.driver.minor=154
nvidia.com/gpu.compute.major=6
nvidia.com/cuda.runtime.minor=2
nvidia.com/cuda.driver.rev=05
nvidia.com/gfd.timestamp=1726648146
nvidia.com/gpu.count=1
nvidia.com/gpu.machine=VMware20,1
nvidia.com/mig.capable=false
nvidia.com/gpu.family=pascal
nvidia.com/gpu.compute.minor=1
nvidia.com/gpu.product=Tesla-P4
nvidia.com/gpu.memory=7680
nvidia.com/cuda.runtime.major=12
```

查看gpu-feature-discovery pod的配置目录

1. 检查配置目录是否有文件
2. 节点是否能够看到对应标签

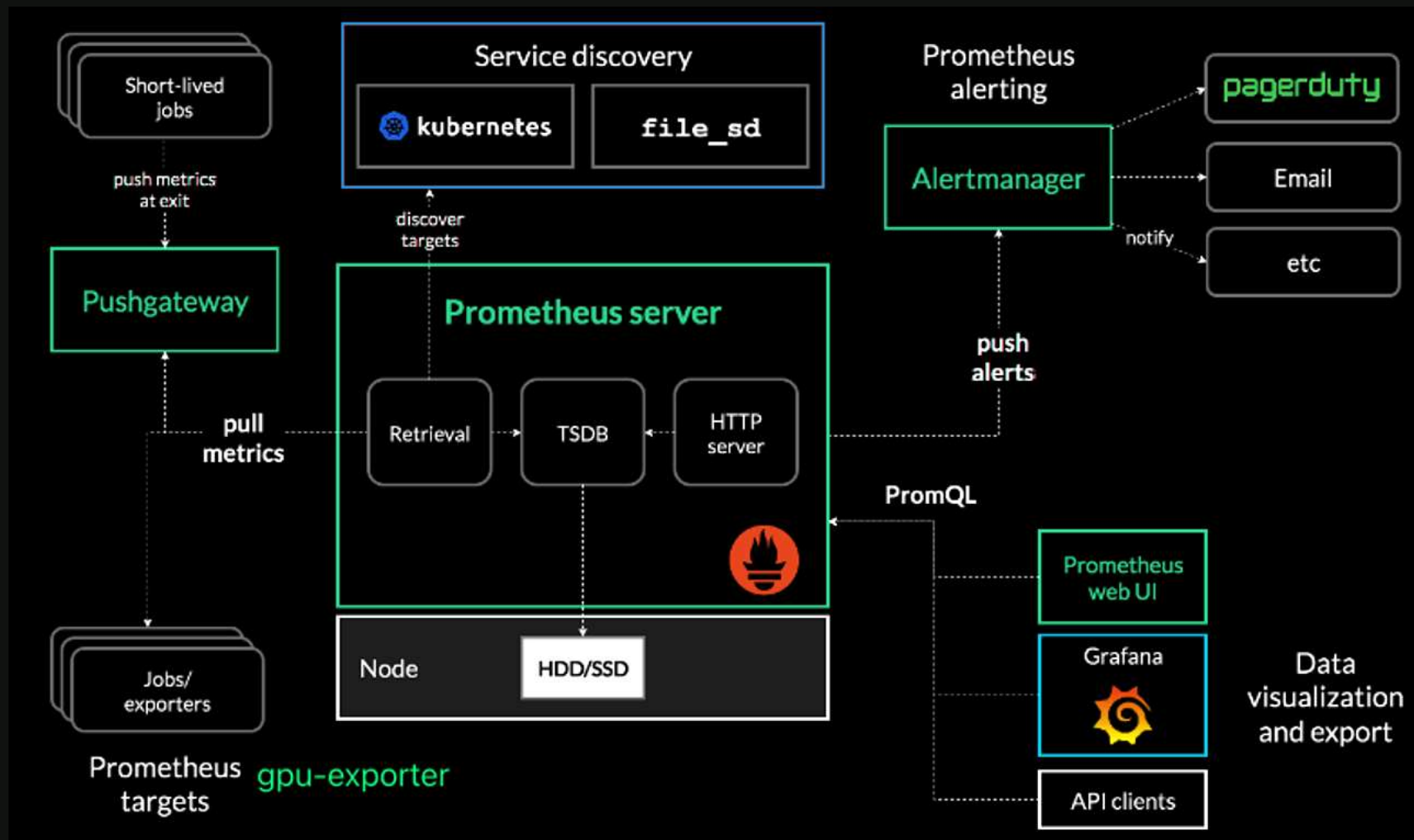
■ dcgm-exporter: gpu 监控数据如何采集的

dcgm-exporter 暴露监控指标, 并通过 ServiceMonitor 发现自己

<https://github.com/NVIDIA/dcgm-exporter>

<https://github.com/NVIDIA/gpu-exporter>

<https://juejin.cn/post/6942457366482780191>



■ Validator: GPU operator 安装自检功能

检查安装是否成功

← workload-002

集群概览

节点管理

工作负载

— 无状态负载

— 有状态负载

— 守护进程

— 任务

— 定时任务

— 容器组

— ReplicaSet

Helm 应用

Operator 应用

容器网络

自定义资源

集群: workload-002 / 命名空间: gpu-operator / 容器组 / nvidia-operator-validator-4wv5g

控制台 监控 日志 ...

● 运行中

状态

0

重启次数

gpu-operator

命名空间

不限制 / 不限制

CPU 请求值/限制值

10.233.102.214

容器组 IP

不限制 / 不限制

内存请求值/限制值

m-10-17-16-19

节点

2024-09-18 16:27

创建时间

容器

标签与注解

事件列表

容器名称	容器类型	状态	镜像	创建时间
nvidia-operator-validator	工作容器	● 运行中	10.17.16.10/nvcr.m.daocloud.io/nvidia/c...	2024-09-18 16:29
plugin-validation	初始化容器	● 已终止	10.17.16.10/nvcr.m.daocloud.io/nvidia/c...	2024-09-18 16:29
cuda-validation	初始化容器	● 已终止	10.17.16.10/nvcr.m.daocloud.io/nvidia/c...	2024-09-18 16:29
toolkit-validation	初始化容器	● 已终止	10.17.16.10/nvcr.m.daocloud.io/nvidia/c...	2024-09-18 16:28
driver-validation	初始化容器	● 已终止	10.17.16.10/nvcr.m.daocloud.io/nvidia/c...	2024-09-18 16:28

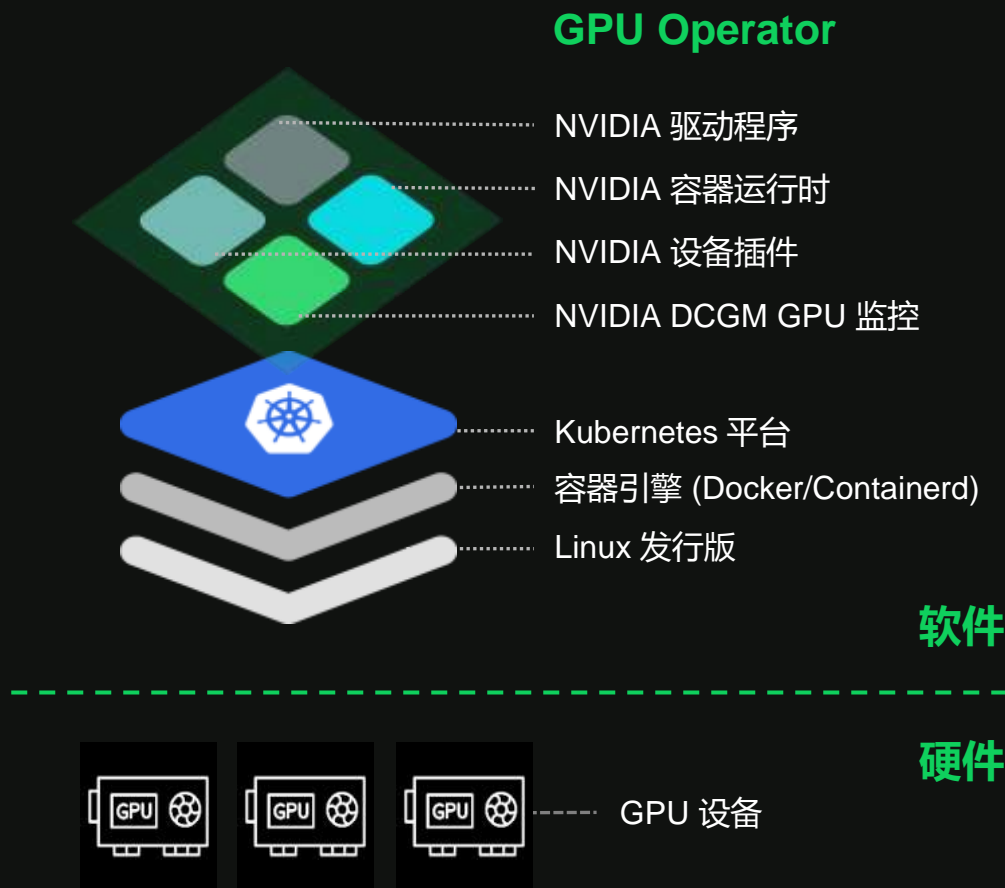
● 运行中

Powered By DaoCloud

■ 这次只聊 NVIDIA GPU Operator 🤗

GPU Operator - 自动管理并提供 GPU 所需的所有 NVIDIA 软件组件

- **NVIDIA 设备插件** - 通过设备插件机制将 GPU 公开给 Kubelet
- **NVIDIA 容器工具包** : 实现容器化环境中与 GPU 进行交互
- **GPU 驱动程序** : Nvidia 驱动程序组件允许从容器进行驱动安装
- **NVIDIA GPU 功能发现** : 检测并标记启用 GPU 的节点
- **NVIDIA DCGM GPU 监控** : 采集 GPU 指标



■ 安装完成后的效果

The screenshot displays the DaoCloud management console for a cluster named 'workload-002' in the 'gpu-operator' namespace. The left sidebar shows the navigation menu with '工作负载' (Workloads) selected. The main panel shows the '守护进程' (Daemonset) tab, listing several NVIDIA-related daemonsets. The 'nvidia-operator-validator' is highlighted with a red box, and a green box with an arrow points to it from the right.

集群: workload-002 / 命名空间: gpu-operator / 守护进程

守护进程

输入工作负载名称搜索

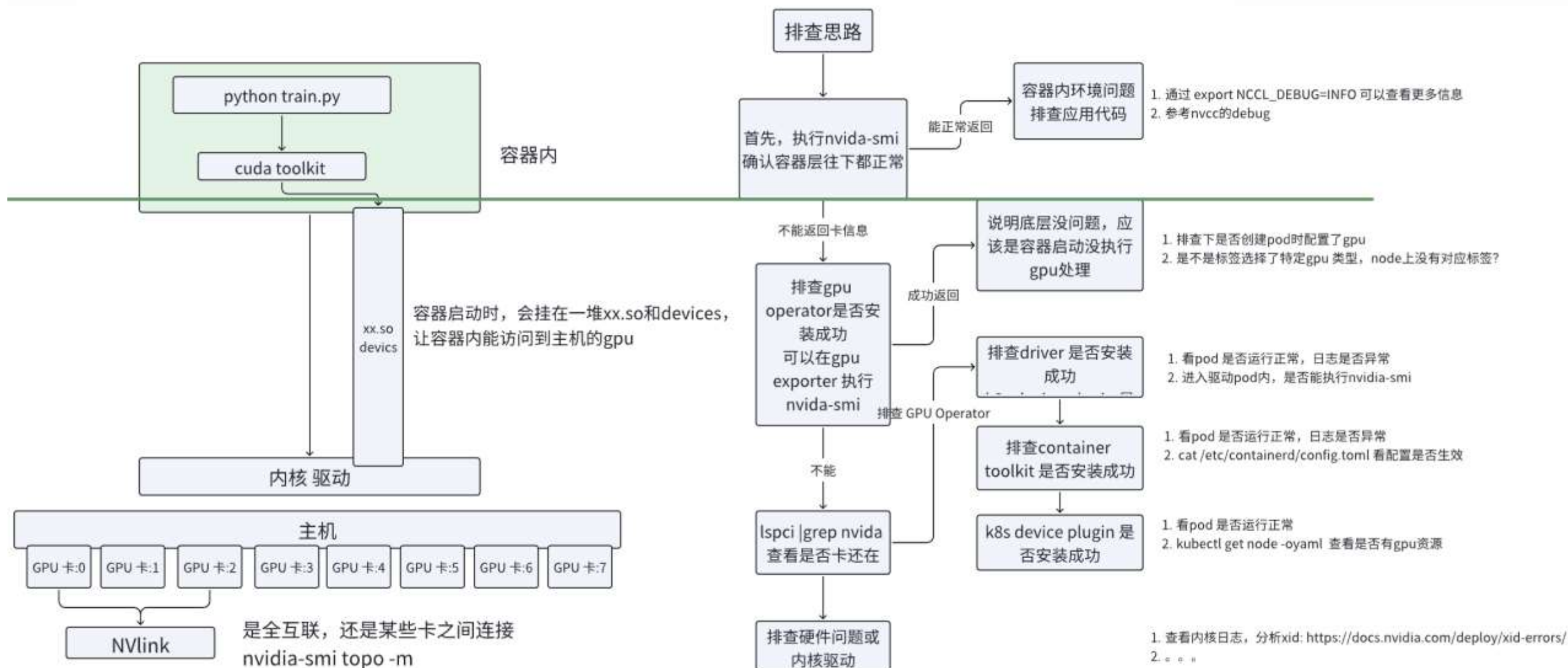
工作负载名称	工作负载别名	状态
<input type="checkbox"/> nvidia-driver-daemonset-5.15.0-94-generic-ubuntu22.04	-	运行中
<input type="checkbox"/> gpu-feature-discovery	-	运行中
<input type="checkbox"/> nvidia-container-toolkit-daemonset	-	运行中
<input type="checkbox"/> nvidia-dcgm-exporter	-	运行中
<input type="checkbox"/> nvidia-device-plugin-daemonset	-	运行中
<input type="checkbox"/> nvidia-mig-manager	-	运行中
<input type="checkbox"/> nvidia-operator-validator	-	运行中
<input type="checkbox"/> gpu-operator-node-feature-discovery-worker	-	运行中

共 8 项

Part 04

故障排查指南&常见故障

故障排查 基本思路



■ 常见故障

问题描述	原因	解决
GPU pod 无法启动	原因比较多，要排查对应的报错原因，但一般报错信息都比较明确	根据pod的事件报错信息，进行对应的排查分析，常见： 1.gpu卡所在节点cpu，内存不足 2.驱动安装有问题或重启后异常 可以挂在内存类型的emptyDir 解决
运行提示/dev/shm 满或太小 [WARNING] /dev/shm size might be too small, if running in docker increase to at least --shm-size='1gb'	gpu 在进场通讯时会使用/dev/shm，默认容器配置的空间比较小	<pre>1 spec: 2 volumes: 3 - name: dshm 4 emptyDir: 5 medium: Memory 6 containers: 7 - image: gcr.io/project/image 8 volumeMounts: 9 - mountPath: /dev/shm 10 name: dshm</pre>
Driver 安装需要 yum 或 apt 下载包	如果不是预编译模式，驱动安装过程中会进行联网进行内核模块编译	新版本支持预编译模式
驱动pod 重启后，会重新卸载安装gpu驱动，导致节点gpu 很长时间不可用	社区一致问题，主要原因是驱动安装是通过把容器内/ 挂在到主机目录实现的，pod重启后挂载文件不完整，只能重新安装再挂载 https://github.com/NVIDIA/gpu-operator/issues/705	如果确实需要优化，可以临时通过在主机上安装驱动方式绕行

Thanks.



扫描二维码，添加我的企业微信