128Kubernetes 系列（一二二）通过 NVIDIA Multi-Instance，每个 GPU 运行更多 Pod

机器学习 (ML) 工作负载需要大量的计算能力。在可扩展的机器学习应用程序所需的所有基础设施组件中，GPU 是最关键的。 GPU 凭借其并行处理能力，彻底改变了深度学习、科学模拟和高性能计算等领域。但并非所有机器学习工作负载都需要相同数量的资源。传统上，机器学习科学家必须为完整的 GPU 付费，无论他们是否需要。

2020 年，NVIDIA 推出了多实例 GPU (MIG) 共享。此功能将 GPU 划分为多个更小的、完全隔离的 GPU 实例。它对于 GPU 计算能力未完全饱和的工作负载特别有利。它允许用户在单个 GPU 上并行运行多个工作负载，以最大限度地提高资源利用率。本文介绍如何在 Amazon EKS 上使用 MIG。

NVIDIA Multi-Instance GPU

MIG 是基于 NVIDIA Ampere 架构的 NVIDIA GPU 的一项功能。它可以让您最大限度地发挥 NVIDIA GPU 的价值并减少资源浪费。使用 MIG，您可以将 GPU 划分为更小的 GPU 实例，称为 MIG 设备。每个 MIG 设备都完全隔离，具有自己的高带宽内存、缓存和计算核心。您可以创建切片来控制每个 MIG 设备的内存量和计算资源数量。

MIG 使您能够微调工作负载获得的 GPU 资源量。此功能提供有保证的服务质量 (QoS) 以及确定的延迟和吞吐量，以确保工作负载可以安全地共享 GPU 资源而不会受到干扰。

NVIDIA 有大量文档(https://docs.nvidia.com/datacenter/tesla/mig-user-guide/?ref=blog.realvarez.com)解释了 MIG 的内部工作原理，因此我不会在这里重复这些信息。

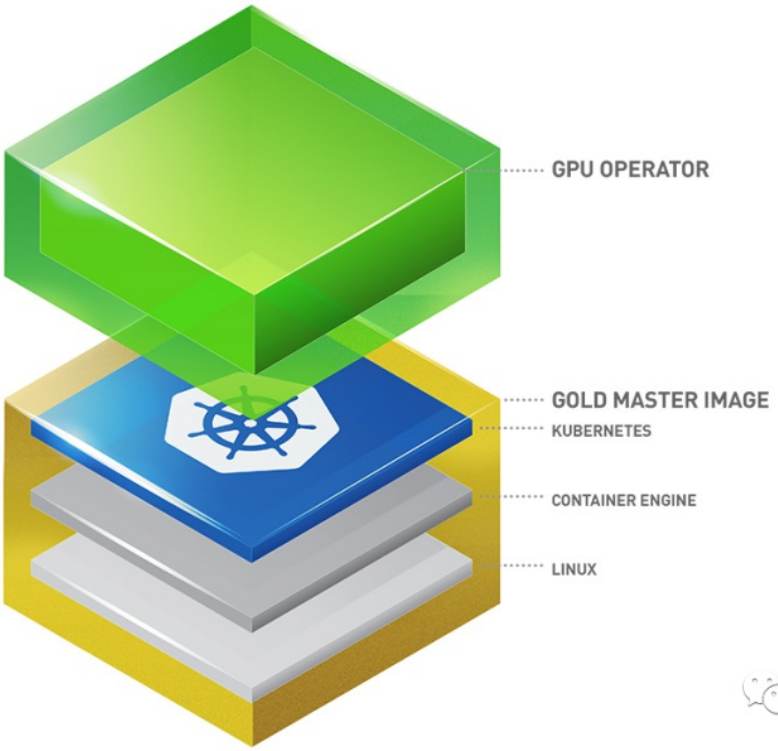
将 MIG 与 Kubernetes 结合使用

我们的客户都选择 Kubernetes 来操作他们的 ML 工作负载。 Kubernetes 提供了强大且可扩展的调度机制，使得在虚拟机集群上编排工作负载变得更加容易。Kubernetes 还拥有充满活力的社区构建工具，例如 Kubeflow，可以更轻松地构建、部署和管理 ML 管道。

由于其复杂性，Kubernetes 上的 MIG 仍然是一个未被充分利用的功能。这里部分要归咎于 NVIDIA 文档。虽然 NVIDIA 的文档解释了 MIG 如何广泛工作（尽管有很多重复），但在提供 Kubernetes 上的 MIG 部署和配置的教程和示例等资源方面却缺乏。更糟糕的是，要在 Kubernetes 上使用 MIG，您必须安装一堆资源，例如 NVIDIA 驱动程序、NVIDIA 容器运行时和设备插件。

值得庆幸的是，NVIDIA GPU Operator（https://github.com/NVIDIA/gpu-operator） 可以自动部署、配置和监控 Kubernetes 中的 GPU 资源。它简化了在 Kubernetes 上使用 MIG 所需组件的安装。其主要特点是：

- 自动GPU驱动安装和管理
- 自动GPU资源分配和调度
- 自动GPU资源分配和调度
- 支持 NVIDIA 容器运行时
- 支持 NVIDIA Multi-Instance GPU (MIG)



Operator 将安装以下组件：

- NVIDIA device driver
- Node Feature Discovery: 检测节点上的硬件功能
- GPU 功能发现：自动为节点上可用的 GPU 集生成标签
- NVIDIA DCGM Exporter: 利用 NVIDIA DCGM 为 Prometheus 公开 GPU 指标 Exporter
- Device Plugin: 公开集群每个节点上的 GPU 数量，跟踪 GPU 的运行状况，并在 Kubernetes 集群中运行支持 GPU 的容器
- Device Plugin Validator: 通过 InitContainers 对每个组件运行一系列验证，并在 /run/nvidia/validations 下写出结果
- NVIDIA 容器工具包 （https://github.com/NVIDIA/nvidia-container-toolkit）
- NVIDIA CUDA Validator
- NVIDIA Operator Validator: 验证驱动程序、工具包、CDA 和 NVIDIA 设备插件
- NVIDIA MIG 管理器:适用于 Kubernetes 集群中 NVIDIA GPU 的 MIG 分区编辑器

```
realvarez @i-0e735c6f200094bba ~/projects/p4d ♥ 22:14 kubectl exec mig3-20-68cd75587c-wjvfz -ti -- nvidia-smi
Tue May 23 22:14:49 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.182.03   Driver Version: 470.182.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  Off  | 00000000:10:1C.0 Off |                   On |
| N/A   42C    P0    74W / 400W |                  N/A |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared       |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                      |
|==================+======================+===========+======================|
|  0    2   0   0  |     10MiB / 20096MiB | 42      0 |  3   0    2    0    0 |
|                  |      0MiB / 32767MiB |           |                      |
+------------------+----------------------+-----------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Amazon EKS 上的 NVIDIA GPU Operator

虽然 NVIDIA GPU Operator 可以轻松地在 Kubernetes 中使用 GPU，但其某些组件需要更新版本的 Linux 内核和操作系统。 Amazon EKS 为 GPU 工作负载提供 Linux AMI，预安装 NVIDIA 驱动程序和容器运行时。在撰写本文时，此 AMI 提供 Linux 内核 5.4。但是，NVIDIA GPU Operator Helm Charts 默认配置为 Ubuntu 或 Centos 8。因此，让 NVIDIA GPU Operator 在 Amazon EKS 上工作并不像执行那么简单：

```
helm install gpu-operator nvidia/gpu-operator
```

演练

让我们从安装 NVIDIA GPU Operator 开始演练。您需要一个 EKS 集群，其节点组由配备 NVIDIA GPU 的 EC2 实例（P4、P3 和 G4 实例）组成。如果您想为此演练创建一个新集群，可以使用以下 eksctl 清单：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: p4d-cluster
  region: eu-west-1
managedNodeGroups:
  - name: demo-gpu-workers
    instanceType: p4d.24xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    volumeSize: 200
```

我将在本演示中使用 P4d.24XL 实例。每个 P4d.24XL EC2 实例都有 8 个 NVIDIA A100 Tensor 核心 GPU。每个 A100 GPU 拥有 40GB 内存。默认情况下，每个 GPU 只能运行一个 GPU 工作负载，每个 Pod 获得 40GB GPU 内存片。这意味着每个实例只能运行 8 个 Pod。

使用 MIG，您可以对每个 GPU 进行分区，以便每个 GPU 运行多个 Pod。在具有 8 个 A100 GPU 的 P4d.24XL 节点上，您可以为每个 GPU 创建 7 个 5GB A100 切片。因此，您可以同时运行 7*8 = 56 个 Pod。或者，您可以创建 24 个具有 10GB 切片的 pod，或 16 个具有 20GB 切片的 pod，或 8 个具有 20GB 切片的 pod。

安装 NVIDIA GPU Operator：

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

helm upgrade gpuo \
    nvidia/gpu-operator \
    --set driver.enabled=true \
    --set mig.strategy=mixed \
    --set devicePlugin.enabled=true \
    --set migManager.enabled=true \
    --set migManager.WITH_REBOOT=true \
    --set toolkit.version=v1.13.1-centos7 \
    --set operator.defaultRuntime=containerd \
    --set gfd.version=v0.8.0 \
    --set devicePlugin.version=v0.13.0 \
    --set migManager.default=all-balanced
```

查看GPU Operator创建的资源：

```
$ kubectl get pods
NAME                                          READY   STATUS      RESTARTS   AGE
gpu-feature-discovery-529vf                   1/1     Running     0          20m
gpu-operator-9558bc48-z4wlh                   1/1     Running     0          3d20h
gpuo-node-feature-discovery-master-7f8995bd8b-d6jdj   1/1     Running     0          3d20h
gpuo-node-feature-discovery-worker-wbtxc      1/1     Running     0          20m
nvidia-container-toolkit-daemonset-lmpz8      1/1     Running     0          20m
nvidia-cuda-validator-bxmhj                   0/1     Completed   1          19m
nvidia-dcgm-exporter-v8p8f                    1/1     Running     0          20m
nvidia-device-plugin-daemonset-7ftt4          1/1     Running     0          20m
nvidia-device-plugin-validator-pf6kk          0/1     Completed   0          18m
nvidia-mig-manager-82772                      1/1     Running     0          18m
nvidia-operator-validator-5fh59               1/1     Running     0          20m
```

GPU 功能发现向节点添加标签，帮助 Kubernetes 调度需要 GPU 的工作负载。您可以通过描述节点来查看标签：

```
$ kubectl describe node
...
Allocatable:
  attachable-volumes-aws-ebs:  39
  cpu:                         95690m
  ephemeral-storage:           18242267924
  hugepages-1Gi:               0
  hugepages-2Mi:               0
  memory:                      1167644256Ki
  nvidia.com/gpu:              8
  pods:                        250
...
```

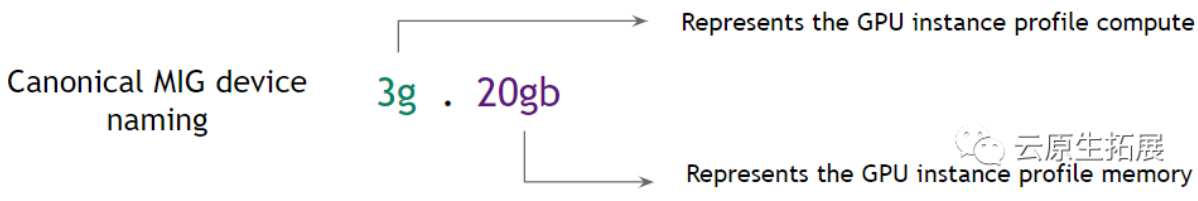
Pod 可以通过在资源中指定 GPU 来请求 GPU。以下是 pod 清单示例：

```
kind: Pod
metadata:
  name: dcgmproftester-1
spec:
  restartPolicy: "Never"
  containers:
  - name: dcgmproftester11
    image: nvidia/samples:dcgmproftester-2.0.10-cuda11.0-ubuntu18.04
    args: ["--no-dcgm-validation", "-t 1004", "-d 30"]
    resources:
      limits:
         nvidia.com/gpu: 1
    securityContext:
      capabilities:
        add: ["SYS_ADMIN"]
```

我们不会创建一个使用完整 GPU 的 pod，因为它可以开箱即用。相反，我们将创建使用部分 GPU 的 Pod。

在 Kubernetes 上创建 MIG 分区

NVIDIA 提供了两种在 Kubernetes 节点上公开 MIG 分区设备的策略。在单一策略中，节点仅在所有 GPU 上公开单一类型的 MIG 设备。而混合策略允许您跨节点的所有 GPU 创建多个不同大小的 MIG 设备。



MIG 设备命名

使用 MIG 单一策略，您可以创建类似大小的 MIG 设备。在 P4d.24XL 上，您可以创建 56 个 1g.5gb 切片、24 个 2g.10gb 切片、16 个 3g.20gb 切片、或者 1 个

4g.40gb 或 7g.40gb 切片。

混合策略将允许您创建一些 1g.5gb 以及一些 2g.10gb 和 3g.20gb 切片。当您的集群的工作负载具有不同的 GPU 资源要求时，它非常有用。

使用单一策略创建 MIG 设备

让我们创建一个策略并看看如何将其与 Kubernetes 一起使用。 NVIDIA GPU Operator 可以轻松创建 MIG 分区。要配置分区，您所要做的就是标记节点。 MIG 管理器在所有节点上作为守护进程运行。当它检测到节点标签时，它将使用 mig-parted 创建 MIG 设备。

标记节点以在所有 GPU 上创建 1g.5gb MIG 设备（将 $NODE 替换为集群中的节点）：

```
kubectl label nodes $NODE nvidia.com/mig.config=all-1g.5gb --overwrite
```

一旦您以这种方式标记节点，就会发生两件事。首先，节点将不再通告任何完整的 GPU，并且 nvidia.com/gpu 标签将设置为 0。其次，您的节点将通告 56 个 1g.5gb MIG 设备。

```
$ kubectl describe node $NODE
...
  nvidia.com/gpu:              0
  nvidia.com/mig-1g.5gb:       56
...
```

请注意，更改可能需要几秒钟才能生效。当更改仍在进行时，该节点将具有标签 `nvidia.com/mig.config.state=pending` 。一旦 MIG 管理器完成分区，标签将设置为 `nvidia.com/mig.config.state=success` 。

我们现在可以创建使用 MIG 设备的部署。

```
cat << EOF > mig-1g-5gb-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mig1.5
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mig1-5
  template:
    metadata:
      labels:
        app: mig1-5
    spec:
      containers:
      - name: vectoradd
        image: nvidia/cuda:8.0-runtime
        command: ["/bin/sh", "-c"]
        args: ["nvidia-smi && tail -f /dev/null"]
        resources:
          limits:
            nvidia.com/mig-1g.5gb: 1
EOF
```

您现在应该有一个正在运行的 pod，它消耗 1x 1g.5gb MIG 设备。

```
$ kubectl get deployments.apps mig1.5
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
mig1.5   1/1     1            1           1h
```

让我们将部署扩展到 100 个副本。将仅创建 56 个 Pod，因为该节点只能容纳 56 个 1g.5gb MIG 设备（8 个 GPU * 每个 GPU 7 个 MIG 切片）。

```
kubectl scale deployment mig1.5 --replicas=100
```

请注意，只有 56 个 Pod 可用：

```
kubectl get deployments.apps mig1.5
NAME     READY    UP-TO-DATE   AVAILABLE   AGE
mig1.5   56/100   100          56          1h
```

执行到容器之一并运行 nvidia-smi 以查看分配的 GPU 资源。

```
kubectl exec <YOUR MIG1.5 POD> -ti -- nvidia-smi
```

```
realvarez @i-0e735c6f200094bba ~/projects/p4d ♥ 20:32 kubectl exec mig1.5-57d9477d55-z44j2 -ti -- nvidia-smi
Tue May 23 20:32:24 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.182.03   Driver Version: 470.182.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  Off  | 00000000:10:1D.0 Off |                   On |
| N/A   39C    P0    74W / 400W |                  N/A |     N/A      Default |
|                               |                      |              Enabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| MIG devices:                                                                |
+------------------+----------------------+-----------+----------------------+
| GPU  GI  CI  MIG |         Memory-Usage |        Vol|         Shared       |
|      ID  ID  Dev |           BAR1-Usage | SM     Unc| CE  ENC  DEC  OFA  JPG|
|                  |                      |        ECC|                      |
|==================+======================+===========+======================|
|  0    7   0   0  |      3MiB /  4864MiB | 14      0 |  1   0    0    0    0 |
|                  |      0MiB /  8191MiB |           |                      |
+------------------+----------------------+-----------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```


正如你所看到的，这个 Pod 只有 5GB 内存。

让我们将部署规模缩小到 0：

```
kubectl scale deployment mig1.5 --replicas=0
```

使用混合策略创建 MIG 设备

在单一策略中，所有 MIG 设备都是 1g.5gb 设备。现在让我们对 GPU 进行切片，以便每个节点都支持多个 MIG 设备配置。 MIG 管理器使用 configmap 来存储 MIG 配置。当我们用 all-1g.5gb 标记节点时，MIG 分区编辑器使用 configmap 来确定分区方案

```
$ kubectl describe configmaps default-mig-parted-config
...

  all-1g.5gb:
    - devices: all
      mig-enabled: true
      mig-devices:
        "1g.5gb": 7

...
```

此配置映射还包括其他配置文件，例如 all-balanced 。 all-balanced 配置文件为每个 GPU 创建 2x 1g.10gb、1x 2g.20gb 和 1x 3g.40gb MIG 设备。您可以通过编辑 configmap 创建自己的自定义配置文件。

all-balanced MIG 简介：

```
$ kubectl describe configmaps default-mig-parted-config

...
  all-balanced:
    - device-filter: ["0x20B010DE", "0x20B110DE", "0x20F110DE", "0x20F610DE"]
      devices: all
      mig-enabled: true
      mig-devices:
        "1g.5gb": 2
        "2g.10gb": 1
        "3g.20gb": 1
...
```

让我们标记节点以使用 all-balanced MIG 配置文件：

```
kubectl label nodes $NODE nvidia.com/mig.config=all-balanced --overwrite
```

一旦节点具有 nvidia.com/mig.config.state=success 标签，描述该节点，您将看到该节点中列出了多个 MIG 设备：

```
$ kubectl describe node $NODE

...

  nvidia.com/mig-1g.5gb:       16
  nvidia.com/mig-2g.10gb:      8
  nvidia.com/mig-3g.20gb:      8

...
```

使用 all-balanced 配置文件，此 P4d.24XL 节点可以运行 16 个 1g.5gb、8 个 2g.20gb 和 8 个 3g.20gb Pod。

让我们通过创建两个额外的部署来测试一下。一个包含使用一个 2g.10gb MIG 设备的 Pod，另一个使用 3g.10gb MIG 设备。

```
cat << EOF > mig-2g-10gb-and-3g.20gb-deployments.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mig2-10
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mig2-10
  template:
    metadata:
      labels:
        app: mig2-10
    spec:
      containers:
      - name: vectoradd
        image: nvidia/cuda:8.0-runtime
        command: ["/bin/sh", "-c"]
        args: ["nvidia-smi && tail -f /dev/null"]
        resources:
          limits:
            nvidia.com/mig-2g.10gb: 1
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mig3-20
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mig3-20
  template:
    metadata:
      labels:
        app: mig3-20
    spec:
      containers:
      - name: vectoradd
        image: nvidia/cuda:8.0-runtime
        command: ["/bin/sh", "-c"]
        args: ["nvidia-smi && tail -f /dev/null"]
        resources:
          limits:
            nvidia.com/mig-3g.20gb: 1
EOF
```

这些部署中的 Pod 运行后，将所有三个部署扩展到 20 个副本：

```
kubectl scale deployments mig1.5 mig2-10 mig3-20 --replicas=20
```

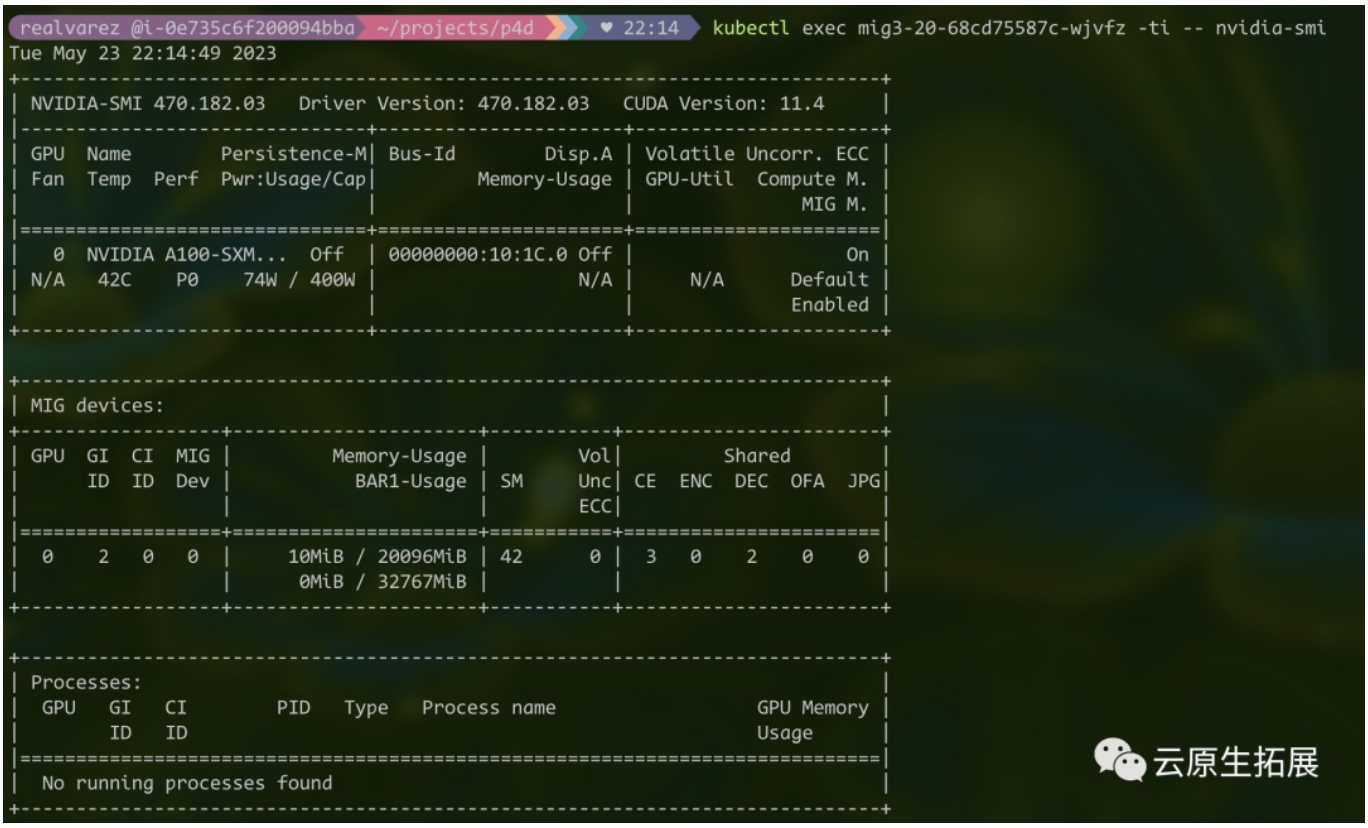
让我们看看有多少副本开始运行：

```
realvarez @i-0e735c6f200094bba ~/projects/p4d ♥ 22:12 kubectl get deployments.apps mig1.5 mig2-10 mig3-20
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
mig1.5     16/20   20           16          8d
mig2-10    8/20    20           8           4m50s
mig3-20    8/20    20           8           3m30s
```


让我们看看 3g.20gb pod 获得了多少 GPU 内存：

```
kubectl exec mig3-20-<pod-id> -ti -- nvidia-smi
```

正如预期的那样，该 Pod 分配了 20GB GPU 内存。



清理

删除集群和节点组：

```
eksctl delete cluster <CLUSTER_NAME>
```

总结

本文介绍如何使用 NVIDIA 多实例 GPU 对 GPU 进行分区并将其与 Amazon EKS 结合使用。在 Kubernetes 上使用 MIG 可能很复杂，但 NVIDIA GPU Operator 简化了安装 MIG 依赖项和分区的过程。

通过利用 MIG 的功能和 NVIDIA GPU Operator 提供的自动化功能，ML 科学家可以优化 GPU 使用，每个 GPU 运行更多工作负载，并在可扩展的 ML 应用程序中实现更好的资源利用率。由于能够在每个 GPU 上运行多个应用程序并定制资源分配，您可以优化 ML 工作负载，以在应用程序中实现更高的可扩展性和性能。