曦云® 系列通用计算 GPU

云原生参考手册

CSOG-24001-020-F3_V05　　2024-09-30

沐曦专有和三级保密信息　　本文档受 NDA 管控

声明

更新记录

版本	日期	更新说明
V05	2024-09-30	更新以下章节： • 设置 Chart 选项 新增以下章节： • 启用 gpu-aware（可选） • gpu-aware
V04	2024-09-06	更新以下章节： • gpu-label 新增以下章节： • MetaX Docker
V03	2024-07-31	更新以下章节： • Helm Chart • 软件清单 • 离线安装包 • GPU Extensions • GPU Operator 新增以下章节： • MXMACA®容器镜像
V02	2024-03-15	新增曦云®系列 GPU 产品信息
V01	2024-01-31	正式版本首次发布

目录

1 概述

Kubernetes（k8s）是一款流行的开源容器编排器，广泛应用于数据中心等场景下。沐曦实现了一系列的基础软件，用于支持在 Kubernetes 上使用沐曦 GPU 以及 MXMACA® 软件栈。这些组件以容器的形式提供，运行在 k8s 集群上，服务于用户应用运行的各个环节。针对于不同的使用需求和场景，沐曦提供了两个名为 GPU Extensions 和 GPU Operator 的 `Helm Chart`，用于 k8s 组件的部署。

1.1 Kubernetes 组件

gpu-device 基于 Kubernetes Device Plugin Framework 实现，负责向 Kubernetes 注册沐曦 GPU 设备，并以 `metax-tech.com/gpu` 资源的形式提供。**gpu-device** 包含了资源分配器的逻辑实现，对于确定份数的 GPU 资源请求，**gpu-device** 总是确保从避免资源碎片，卡间互联拓扑角度进行最优分配。**gpu-device** 会定期检查沐曦 GPU 设备的健康状态，识别出故障的 GPU 资源，并将其从可分配资源中移除。

gpu-label 是沐曦私有实现的控制器，负责监控 k8s 节点上沐曦 GPU 及 MXMACA® 软件栈的状态信息，并以标签的形式对节点进行标记。用户提交任务时，可通过在 `nodeSelector` 字段设置节点标签的形式来筛选符合预期的节点。例如，根据沐曦 GPU 显卡型号、虚拟化配置等要求筛选节点。

driver-manager 负责维护 k8s 节点上的驱动，包括 MXMACA® 用户态软件栈和内核驱动的安装及维护，以及虚拟化相关配置。

container-runtime 沐曦私有容器运行时，是 k8s 云原生方案的核心组件。在 **container-runtime** 的帮助下，用户的应用容器镜像无需携带 MXMACA® 软件栈，因此可以得到尺寸更小的容器镜像，并且在有升级 MXMACA® 版本的需求时也无需重新构建容器镜像。

operator-controller 基于 Operator Framework 实现，负责自动化地完成沐曦 k8s 组件在集群上的部署以及配置工作。

1.2 Helm Chart

沐曦提供两种 k8s 组件的推荐部署方案，并分别制作了 `Helm Chart` 以方便使用，用户应根据实际的应用场景选择其一。

GPU Extensions 方案使用 **gpu-device** 和 **gpu-label** 两个组件，为容器使用沐曦 GPU 提供必需的 GPU 设备资源注册及分配能力。此方案较为简明，对集群环境要求少，要求 Kubernetes 版本 ≥ 1.23。

GPU Operator 方案使用全部的组件，除具备 GPU Extensions 方案的全部能力之外，针对 MXMACA® 软件栈在 Kubernetes 集群中的使用进一步云原生化，减轻集群中软件运行负担，降低运维难度。此方案要求集群满足如下条件：

Kubernetes		⩾ 1.23
Container Runtime	docker	⩾ 18.09.3
	containerd	⩾ 1.5.0
Host OS	Ubuntu	18.04 / 20.04 / 22.04
	CentOS	8.x / 9.x
	OpenEuler	20.03 / 22.03

1.3 软件清单

本文所涉及的软件列表如下，用户可使用 `helm` 或 `docker` 工具，根据下表中提供的 URL 来获取相应制品。

Type	Artifact URL
Helm Chart	oci://cr.metax-tech.com/cloud/metax-gpu-extensions
Helm Chart	oci://cr.metax-tech.com/cloud/metax-operator
Image	cr.metax-tech.com/cloud/gpu-device
Image	cr.metax-tech.com/cloud/gpu-label
Image	cr.metax-tech.com/cloud/driver-manager
Image	cr.metax-tech.com/cloud/container-runtime
Image	cr.metax-tech.com/cloud/operator-controller

注解： 请通过商务途径获得 `cr.metax-tech.com` 的访问权限或离线安装包。

1.4 离线安装包

Kubernetes 组件容器镜像以及 `Helm Chart` 可以从 `metax-gpu-k8s-package.<VERSION>.tar.gz` 压缩包中获取。以 0.8.0 版本为例，解压 `metax-gpu-k8s-package.0.8.0.tar.gz` 以获取容器镜像安装工具，以及 GPU Extensions 和 GPU Operator 的 `Helm Chart`。

```
$ mkdir k8s
$ tar -C k8s -xvzf metax-gpu-k8s-package.0.8.0.tar.gz
metax-k8s-images.0.8.0.run
metax-gpu-extensions-0.8.0.tgz
metax-operator-0.8.0.tgz
```

注解： GPU Operator 目前仅支持曦云®系列加速卡，其他系列的沐曦产品应使用 `metax-gpu-k8s-package.<VERSION>-base.tar.gz` 软件包，其中不会提供 `metax-operator-0.8.0.tgz`。

1.4.1 推送容器镜像

通常，在 Kubernetes 集群网络环境内部需要部署容器仓库服务，以提升运行效率。假设容器仓库服务器域名为 `DOMAIN`，用于存放沐曦 Kubernetes 组件镜像的项目名为 `PROJECT`。集群管理员需按照以下步骤完成镜像的推送：

1. **安装 Docker**

   确认进行操作的主机已正确安装 Docker 并完成用户组配置，确认 Docker 命令可正常工作。安装步骤可参考 Docker 官方文档 。

2. **登录容器仓库，获得推送权限**

```
docker login DOMAIN -u $USER
```

3. **推送容器镜像至仓库**

```
./k8s/metax-k8s-images.0.8.0.run push DOMAIN/PROJECT
```

注解： 在一些规模较小或验证性质的集群中，可能缺失容器仓库服务，管理员可将 `metax-k8s-images.0.8.0.run` 分发至集群的所有节点，并在所有节点上执行如下命令加载容器镜像。

```
./k8s/metax-k8s-images.0.8.0.run load
```

1.4.2 推送 Helm Chart

Helm 是专为 Kubernetes 设计的包管理器，沐曦 Kubernetes 软件方案采用 Helm Chart 的形式发布，以提供最佳的部署体验。沐曦提供 OCI 兼容的 Helm Chart，因此可以选择将 Helm Chart 推送至容器镜像仓库以便后续使用。可参考如下操作步骤：

1. **安装 Helm 工具**

   在 Kubernetes 的管理节点上安装 Helm 工具，可运行如下命令：

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
↪| bash
```

   在一些受限的环境下，也可参考 Helm 官方文档 选择合适的方式进行安装。

注解： Helm 版本要求 ⩾ 3.0。

---

2. **登录容器仓库，获取推送权限**

```
helm registry login DOMAIN -u $USER
```

3. **推送 Helm Chart 至仓库**

```
helm push ./k8s/metax-gpu-extensions-0.8.0.tgz oci://DOMAIN/PROJECT
helm push ./k8s/metax-operator-0.8.0.tgz oci://DOMAIN/PROJECT
```

注解： 在一些规模较小或验证性质的集群中，管理员可直接使用本地 Helm Chart 包进行 Kubernetes 组件安装，但需自行保管好 Chart 包以便后续运维使用。

1.4.3 MXMACA® 容器镜像

MXMACA® 容器镜像压缩包随 MXMACA® 软件栈的发布包发布，需另外获取。根据软件版本的不同，镜像及压缩包命名可能存在一些差异，应以实际获得的离线包为准。用户也可通过商务途径获得授权后，通过 `cr.metax-tech.com/library` 容器镜像仓库获取容器镜像。

- **2.23.0.x 及之前版本**

  在 2.23.0 的部分版本及之前版本的 MXMACA® 软件栈的发布包中，可找到文件名中包含 `-container-` 字段的压缩包。执行如下命令加载容器镜像：

```
$ docker load < maca-c500-container-2.23.0.4-ubuntu18.04-amd64.xz
9a0ca4bf2bcb: Loading layer [=========================>]  46.79MB/
↪46.79MB
134ccdd0eff6: Loading layer [=========================>]  5.095GB/
↪5.095GB
41b261c7229d: Loading layer [=========================>]  27.65kB/
↪27.65kB
Loaded image: cr.metax-tech.com/library/maca-c500:2.23.0.4-
↪ubuntu18.04-amd64
```

  注解： GPU 产品与容器镜像命名之间的对应关系如下：

MXN100	maca-n100:<version>
MXC500	maca-c500:<version>

- **2.23.0.x 及之后版本**

  在 2.23.0 的部分版本及之后版本的 MXMACA® 软件栈的发布包中，可找到扩展名为 `.container.xz` 的压缩包。执行如下命令加载容器镜像：

```
$ docker load < mxc500-maca-2.23.0.4-ubuntu20.04-amd64.container.
↪xz
2d4ccb11c2ef: Loading layer [=========================>]  9.728kB/
↪9.728kB
a3e293fa8e05: Loading layer [=========================>]  1.111MB/
↪1.111MB
c8c5c0b32bb0: Loading layer [=========================>]  15.53MB/
↪15.53MB
447d0cf981f9: Loading layer [=========================>]  9.518GB/
↪9.518GB
Loaded image: mxc500-maca:2.23.0.4-ubuntu20.04-amd64
```

2 GPU Extensions

重要： 在 Kubernetes 集群上配置 GPU Extensions 前，应确认环境满足如下条件：

- Kubernetes 版本 ⩾ 1.23 。
- 管理节点以及工作节点应允许创建特权容器[1] 。
- 工作节点已正确安装 MXMACA® 内核态驱动。安装步骤参见《曦云系列通用计算 GPU 用户指南》相关章节。

2.1 部署参考

GPU Extensions 提供 Helm Chart 形式的安装包，关于 Helm Chart 安装包使用前的准备工作，参见离线安装包 章节。

2.1.1 安装 GPU Extensions

在 Kubernetes 管理节点执行如下命令，即可完成 GPU Extensions 的安装：

```
helm install oci://DOMAIN/PROJECT/metax-gpu-extensions \
  --create-namespace -n metax-gpu-extensions \
  --generate-name \
   --wait \
  --set registry=DOMAIN/PROJECT
```

2.1.2 设置 Chart 选项

GPU Extensions Chart 安装时可通过 `--set option=value` 的方式改变选项默认设置。对于结构体类型选项，配置其成员选项时使用 . 字符连接。例如，指定 `--set gpuLabel.log.format=text` 参数，设置 gpu-label 组件使用 `text` 日志格式。

更多选项，参见下方表格。

[1] kube-apiserver 及 kubelet 应保持配置 `--allow-privileged=true`

表 2.1 Global Options

选项	类型	描述
registry	string	组件容器镜像仓库，默认为 `cr.metax-tech.com/cloud`，通常需指定为集群本地镜像仓库 `DOMAIN/PROJECT`
pullPolicy	enum	镜像拉取策略，默认为 `IfNotPresent`，可选值为 `Always` 和 `Never`，通常无需修改
deployExtenderGPUAware	bool	K8s 调度器扩展，该扩展拥有集群 GPU 资源的拓扑感知能力，可将申请 GPU 的任务调度到最适合的节点。默认为 `false`，可选值为 `true` 和 `false`。在需要使用此扩展时，设置为 `true`
log	struct	日志选项，参考Log Options
gpuDevice	struct	gpu-device 组件选项，参考gpu-device Options
gpuLabel	struct	gpu-label 组件选项，参考gpu-label Options
gpuAware	struct	gpu-aware 组件选项，参考gpu-aware Options

表 2.2 gpu-device Options

选项	类型	描述
healthyInterval	integer	GPU 健康检查间隔的秒数，＞ 0 时启用功能，＝ 0 时禁用功能，默认值为 5
log	struct	日志选项，参考Log Options

表 2.3 gpu-label Options

选项	类型	描述
log	struct	日志选项，参考Log Options

表 2.4 gpu-aware Options

选项	类型	描述
scoreWeight	string	影响任务最终的调度结果。拓展调度器会对各节点进行评分，分数越高越容易被选中。scoreWeight 数值越大，通信效率高的节点获得的分数越高
lossWeight	string	影响任务最终的调度结果。拓展调度器会对各节点进行评分，分数越高越容易被选中。lossWeight 数值越大，调度后对集群拓扑影响小的节点获得的分数越高

表 2.5 Log Options

选项	类型	描述
dir	string	日志文件存储目录，默认为 `/var/log/metax`
level	enum	日志级别，默认为 `info`，可设置值为 `debug`，`info`，`warning`，`error`，`fatal`
format	enum	日志格式，默认为 `json`，可设置值为 `json`，`text`
rotationTime	string	日志轮替时间，可选单位为 `w`(星期)，`d`(日)，`h`(小时)，默认为 1w
maxAge	string	日志保存的最长时间，可选单位为 `w`(星期)，`d`(日)，`h`(小时)，默认为 26w

2.1.3 验证部署

在运行 `helm install` 命令并等待其正确退出后，可通过如下方式确认节点状态正确：

- **GPU 资源数量正确**

执行 kubectl describe node <NODE> 检查工作节点上的 metax-tech.com/gpu 资源数量。

```
$ kubectl describe node <NODE>
...
Capacity:
  cpu:                16
  ephemeral-storage:  143245508Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             16398052Ki
  metax-tech.com/gpu: 4
  pods:               110
Allocatable:
  cpu:                16
  ephemeral-storage:  132015059955
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             16398052Ki
  metax-tech.com/gpu: 4
  pods:               110
...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource            Requests    Limits
  --------            --------    ------
  cpu                 100m (2%)   100m (2%)
  memory              50Mi (0%)   50Mi (0%)
  ephemeral-storage   0 (0%)      0 (0%)
  hugepages-1Gi       0 (0%)      0 (0%)
  hugepages-2Mi       0 (0%)      0 (0%)
  metax-tech.com/gpu  0           0
...
```

metax-tech.com/gpu 资源数量有三处呈现，分别代表不同的涵义：

- Capacity 为节点上沐曦 GPU 设备文件数量。

  未开启虚拟化时与 GPU 物理设备数量相等；节点开启虚拟化后，预期资源数量为 $N_{(gpu)} \times M_{(vfnum)}$。

- Allocatable 为节点上健康资源数量。

  节点状态正常时与 Capacity 保持一致；当 GPU 发生故障时，故障 GPU 所对应的资源数量会在此字段做相应减除，而 Capacity 中仍然保持不变。

- Allocated resources 为当前已分配资源数量。

- **标签状态正确**

执行 kubectl describe node <NODE> 检查工作节点上 metax-tech.com 域下的标签状态。

```
$kubectl describe node <NODE>
...
Labels:       beta.kubernetes.io/arch=amd64
              ...
              metax-tech.com/driver.ready=true
              metax-tech.com/gpu.driver.major=2
              metax-tech.com/gpu.driver.minor=0
              metax-tech.com/gpu.driver.mode=native
              metax-tech.com/gpu.driver.patch=0
              metax-tech.com/gpu.family=MXC
```

(下页继续)

(续上页)

```
                metax-tech.com/gpu.installed=true
                metax-tech.com/gpu.memory=64GB
                metax-tech.com/gpu.product=C550
                metax-tech.com/gpu.vfnum=0
                ...
```

重点确认以下标签状态是否与实际情况相符：

- `metax-tech.com/gpu.installed=true` 表明节点上配置了沐曦 GPU，若为 `false`，则 `metax-tech.com` 域下其余标签状态可忽略。
- `metax-tech.com/gpu.product` 表明节点上配置的沐曦 GPU 产品型号。
- `metax-tech.com/gpu.vfnum` 表明节点上的虚拟化配置。未开启虚拟化时，值为 0；开启虚拟化后，根据不同产品的能力，可能为 1，2，4，8 。

2.1.4 卸载 GPU Extensions

若需卸载 GPU Extensions，可先通过 `helm list -n metax-gpu-extensions` 命令查询已安装的 Chart，再执行 `helm uninstall <CHART> -n metax-gpu-extensions --wait` 命令进行卸载。

注解： `-n` 指定的参数为 `namespace`，应与安装时所指定的值保持一致。

也可直接运行以下脚本自动检查已安装的 GPU Extensions 并进行卸载。

```
chart=$(helm list -q -f "metax" -n metax-gpu-extensions)
if [[ -n $chart ]]; then
    helm uninstall $chart -n metax-gpu-extensions --wait
fi
```

2.1.5 启用 gpu-aware （可选）

如需使用 gpu-aware，应在安装 GPU Extensions 时手动添加配置 `--set deployExtenderGPUAware=true`，具体参见部署参考 。

1. **查询 gpu-aware 服务的地址**

   执行 `kubectl get svc -n metax-gpu | grep gpu-aware | awk '{print $3}'` 获取集群中 `gpu-aware` 服务的地址。

2. **为 Kubernetes Scheduler 启用 SchedulerExtender 准备相关的配置信息**

   为了启用 SchedulerExtender，需要在调度器启动时，传入 SchedulerExtender 相关配置信息。在所有管理节点上，新建目录 `/etc/metax` 保存 SchedulerExtender 配置信息。

   ```
   mkdir -p /etc/metax
   touch /etc/metax/extender.yaml
   ```

   SchedulerExtender 配置信息如下所示：

   ```
   cat >> /etc/metax/extender.yaml << EOF
   apiVersion: kubescheduler.config.k8s.io/v1beta3
   kind: KubeSchedulerConfiguration
   clientConnection:
     kubeconfig: "/etc/kubernetes/scheduler.conf"
   extenders:
   ```

(下页继续)

(续上页)

```
- urlPrefix: "http://<gpuAwareSVCIP>:8080"
  enableHTTPS: false
  filterVerb: "filter"
  prioritizeVerb: "prioritize"
  nodeCacheCapable: false
  ignorable: false
  weight: 500
EOF
```

<gpuAwareSVCIP> 代表 gpu-aware 服务的地址，可以使用 `kubectl get svc -n metax-gpu | grep gpu-aware | awk '{print $3}'` 获取。

表 2.6 Extender Options

选项	描述
urlPrefix	访问 Extender 的地址
enableHTTPS	是否使用 HTTPS 和 Extender 通讯
filterVerb	过滤器接口
prioritizeVerb	评分器接口
nodeCacheCapable	每次请求是否需要携带所有信息
ignorable	是否忽略错误
weight	评分器的权重。在启用拓扑感知调度器拓展后，经历调度任务时，会将拓扑感知调度器对节点的打分，扩大权重的倍数，增加拓扑感知调度器拓展对最终调度评分的影响（为了使拓扑感知调度器生效，该权重需要远超其余调度器）

3. **修改 Kubernetes Scheduler 启动参数，启用 SchedulerExtender**

Kubernetes Scheduler 关于 SchedulerExtender 配置的默认配置文件是 `/etc/kubernetes/manifests/kube-scheduler.yaml`。需要修改以下高亮内容：

```
...
spec:
  containers:
  - command:
    - kube-scheduler
    - --authentication-kubeconfig=/etc/kubernetes/scheduler.conf
    - --authorization-kubeconfig=/etc/kubernetes/scheduler.conf
    - --config=/etc/metax/extender.yaml
...
    volumeMounts:
    - mountPath: /etc/kubernetes/scheduler.conf
      name: kubeconfig
      readOnly: true
    - mountPath: /etc/metax/extender.yaml
      name: extenderconfig
      readOnly: true
    ...
  volumes:
  - hostPath:
      path: /etc/kubernetes/scheduler.conf
      type: FileOrCreate
    name: kubeconfig
  - hostPath:
      path: /etc/metax/extender.yaml
      type: File
    name: extenderconfig
```

4. **修改权重**

   权重参数描述参见gpu-aware Options。

```
kubectl edit cm -n metax-gpu $(kubectl get cm -n metax-gpu | grep gpu-
↪aware-plugin | awk '{print $2}')
```

5. **停用 gpu-aware**

   停用 SchedulerExtender，需要删除 SchedulerExtender 配置，需要修改以下高亮内容：

```
...
spec:
  containers:
  - command:
    - kube-scheduler
    - --authentication-kubeconfig=/etc/kubernetes/scheduler.conf
    - --authorization-kubeconfig=/etc/kubernetes/scheduler.conf
    - --config=/etc/metax/extender.yaml
...
    volumeMounts:
    - mountPath: /etc/kubernetes/scheduler.conf
      name: kubeconfig
      readOnly: true
    - mountPath: /etc/metax/extender.yaml
      name: extenderconfig
      readOnly: true
    ...
  volumes:
  - hostPath:
      path: /etc/kubernetes/scheduler.conf
      type: FileOrCreate
    name: kubeconfig
  - hostPath:
      path: /etc/metax/extender.yaml
      type: File
    name: extenderconfig
```

2.2 组件功能

2.2.1 gpu-device

gpu-device 是沐曦基于 Kubernetes Device Plugin Framework[2] 的实现，以容器形式发布并以 `DaemonSet` 的形式部署在 Kubernetes 集群上。该组件除了具备将沐曦 GPU 设备注册为 `metax-tech.com/gpu` 资源这一 Device Plugin 常见功能外，还实现了如下功能：

- **健康检查**

  gpu-device 以一定间隔[3] 周期性检查节点上的每个 GPU 设备，对发现故障的 GPU 资源进行状态上报。Kubernetes 根据上报信息将相关资源从 Allocatable 资源中移除，从而避免故障资源影响后续作业。

- **GPU 拓扑最优分配**

  在单台服务器上配置多张 GPU 时，GPU 卡间根据双方是否连接在相同的 PCIe Switch 或 MetaXLink 下，存在近远（带宽高低）关系。服务器上所有卡间据此形成一张拓扑，如下图所示。

[2] https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
[3] 受 Helm Chart 选项 gpuDevice.healthyInterval 控制

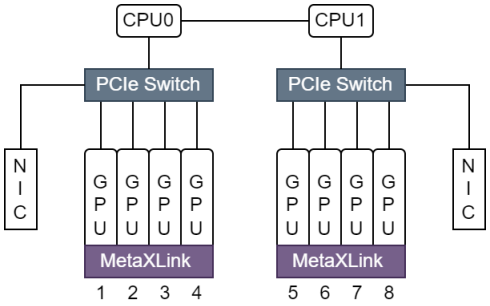

图 2.1 典型拓扑示意图

用户作业请求一定数量的 `metax-tech.com/gpu` 资源，Kubernetes 选择剩余资源数量满足要求的节点，并将 Pod 调度到相应节点。gpu-device 进一步处理资源节点上剩余资源的分配逻辑，并按照以下优先级逻辑为作业容器分配 GPU 设备：

1. MetaXLink 优先级高于 PCIe Switch，包含两层含义：
   - 两卡之间同时存在 MetaXLink 连接以及 PCIe Switch 连接时，认定为 MetaXLink 连接。
   - 服务器剩余 GPU 资源中 MetaXLink 互联资源与 PCIe Switch 互联资源均能满足作业请求时，分配 MetaXLink 互联资源。
2. 分配 GPU 资源尽可能位于相同 MetaXLink 或 PCIe Switch 下。

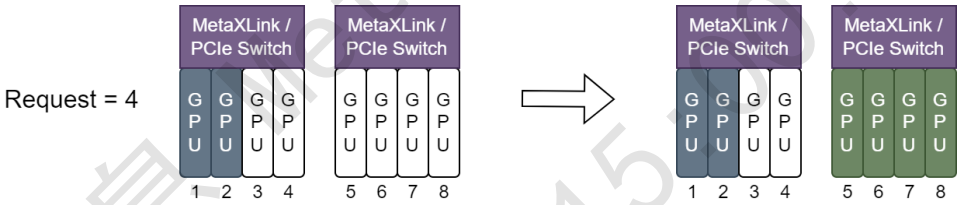

图 2.2 GPU 资源分配示例 1

如上图所示，GPU 1，2 已被分配，新分配 4 张 GPU 时，只有 GPU 5，6，7，8 满足本条规则，因此是唯一分配方案。

3. 分配 GPU 资源后，剩余资源尽可能完整。

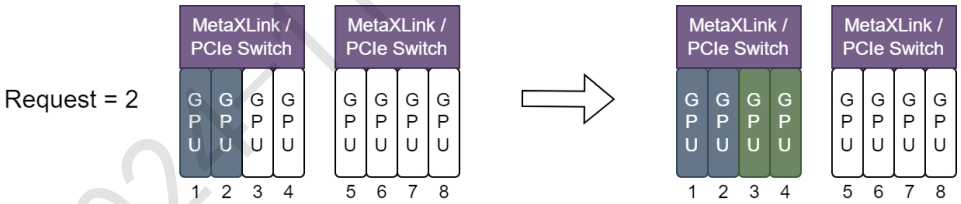

图 2.3 GPU 资源分配示例 2

如上图所示，GPU 1，2 已被分配，新分配 2 张 GPU 时，分配 GPU 3，4 可使剩余资源全部位于同一 MetaXLink 或 PCIe Switch 下，因此是遵循本条规则的唯一分配方案。

注解： 尽管 gpu-device 在拓扑最优以及避免碎片上做了最大的努力，实际使用过程中并不能完全杜绝碎片的产生。因此建议在集群使用时尽量以 1，2，4，8 这样的粒度来分配 GPU，最大限度地降低碎片的可能。

2.2.2 gpu-label

gpu-label 同样以容器的形式发布，并以 DaemonSet 的形式部署在 Kubernetes 集群上。该组件运行时，将检查节点上配置沐曦 GPU 的状态，并在确认配置了沐曦 GPU 的节点上，创建以下标签来描述 GPU 信息以及驱动软件信息。

表 2.7 Labels Under metax-tech.com

标签	类型	描述
gpu.installed	bool	若节点配置了沐曦 GPU，值为 true，否则为 false。若值为 false，则下列所有标签均不会被创建
driver.ready	bool	若节点已安装内核驱动，值为 true，否则为 false
gpu.driver.major	integer	内核驱动程序的主版本号
gpu.driver.minor	integer	内核驱动程序的次版本号
gpu.driver.mode	enum	驱动管理模式，native 表示 Host 上由管理员维护，cloud 表示驱动由容器安装管理
gpu.driver.patch	integer	内核驱动程序的补丁版本号
gpu.family	string	显卡系列
gpu.memory	string	显卡内存大小
gpu.product	string	显卡型号
gpu.sriov	string	SR-IOV 状态，可能为 on、off 或 forbidden
gpu.vfnum	integer	显卡虚拟化配置

2.2.3 gpu-aware

gpu-aware 是沐曦基于 Kubernetes SchedulerExtender 的实现，以容器形式发布并以 DaemonSet 的形式部署在 Kubernetes 集群的管理节点上。该组件的主要功能是，在 Kubernetes 上提交申请 GPU 资源的任务时，会根据感知到的集群 GPU 拓扑信息，结合配置参数的权重进行计算，选出本次调度最合适的节点。

2.3 提交作业

2.3.1 制作容器镜像

1. **加载基础镜像**

   首先需准备镜像，参考MXMACA® 容器镜像 章节加载镜像。

2. **准备 Dockerfile**

   为了得到尺寸精简的应用容器镜像，推荐在 Dockerfile 使用 multi-stage build 功能。以下为 Dockerfile 示例：

```
FROM cr.metax-tech.com/library/maca-c500:2.18.0.3-ubuntu18.04-amd64 as builder

RUN apt-get update && apt-get install -y \
  autoconf \
  build-essential \
  libtool \
  patchelf \
  pkg-config \
  ...

```

(下页继续)

(续上页)

```
11 ENV VAR1=VALUE1
12 ENV VAR2=VALUE2
13 ...
14
15 WORKDIR /workspace
16 COPY . /workspace
17 RUN make OUTPUT=/app
18
19 FROM cr.metax-tech.com/library/maca-c500:2.18.0.3-ubuntu18.04-
   ↪amd64
20
21 RUN apt-get update && apt-get install -y \
22   ... \
23 && rm -rf /var/lib/apt/lists/*
24
25 COPY --from=builder /app /app
```

用户基于此示例编写适用于实际工程的 Dockerfile 时，需注意以下细节：

- [line 1] 使用 MXMACA® 基础镜像作为 `builder` 镜像，提供必要的 MXMACA® 软件栈工具及库文件。
- [line 9] 此处增加构建用户应用所必须的工具和开发依赖库。
- [line 13] 此处设置构建过程所需的环境变量。
- [line 22] 此处安装用户应用运行时依赖库
- [line 25] 从 `builder` 容器中拷贝构建好的应用至基础容器

使用此过程构建应用容器可避免因开发工具、开发依赖库以及构建过程临时文件造成的存储浪费。

3. **构建镜像**

假设用户工作目录有如下结构，Project 为应用项目根目录，Dockerfile 放置于同级目录。

```
workspace
├── Dockerfile
└── Project
    ├── Makefile
    └── src
```

用户可在 workspace 目录运行如下命令完成镜像的构建：

```
docker build -f Dockerfile -t DOMAIN/PROJECT/application:1.0 ./Project
```

2.3.2 准备作业 yaml 文件

用户可参考如下示例编写作业 yaml 文件：

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: application
5    labels:
6      app: application
7  spec:
8    nodeSelector:
9      metax-tech.com/gpu.vfnum: "0"
10     metax-tech.com/gpu.product: "C500"
```

(下页继续)

(续上页)

```
  containers:
  - image: DOMAIN/PROJECT/application:1.0
    name: application-container
    command: ["/bin/bash", "-c", "--"]
    args: ["/app/bin/xxx --option=val; sleep infinity"]
    resources:
      limits:
        metax-tech.com/gpu: 2
```

2.3.3 提交作业

执行 `kubectl create -f sample.yaml` 命令提交作业。

2.4 节点维护

在集群的正常运维过程中，管理员可能需要对工作节点内核驱动进行重装、升级等操作，须遵循以下操作步骤：

1. 使用 `kubectl cordon` 命令停止向节点调度新的任务，移除所有 `metax-tech.com` 域下标签。

```
kubectl cordon <nodename>
```

2. 等待 gpu-device Pod 从节点上退出。
3. 终止所有正在运行的使用沐曦 GPU 的用户任务 Pod。
4. 进行内核驱动卸载、重装、升级等相关操作。
5. 使用 `kubectl uncordon` 命令恢复节点功能。

```
kubectl uncordon <nodename>
```

6. 确认节点标签恢复，节点上可分配资源恢复。

```
kubectl describe node <nodename>
```

3 GPU Operator

重要： 在 Kubernetes 集群上配置 GPU Operator 前，应确认环境满足如下条件：

- Kubernetes 版本及 CRI 支持范围：

Kubernetes	CRI	Version
= 1.23	docker	⩾ 18.09.3
	containerd	⩾ 1.5.0
⩾ 1.24	containerd	⩾ 1.5.0

- 管理节点以及工作节点应允许创建特权容器[1]。
- Host 应使用如下操作系统或其衍生版本，并选择有 MetaX 内核驱动支持的内核版本：

Host OS	Ubuntu	18.04 / 20.04 / 22.04
	CentOS	8.x / 9.x
	OpenEuler	20.03 / 22.03

- 无需且不推荐在工作节点预安装 MXMACA® 内核态驱动，若已安装也无需卸载。

3.1 关于 GPU Operator

GPU Operator 是更加云原生化的方案，除了具备 GPU Extensions 方案的全部功能之外，针对于 MXMACA® 软件栈在 Kubernetes 集群上的部署及应用，进行了多处优化：

- **用户容器镜像优化**

  在使用 GPU Extensions 方案时，用户的应用容器镜像需要基于 MXMACA® 基础镜像构建。而使用 GPU Operator 方案后，用户可选择制作不包含 MXMACA® SDK 的应用容器镜像。如此，可带来两项好处：

  - 镜像尺寸减小
  - MXMACA® 版本升级时无需重新构建应用镜像

  MXMACA® 由 GPU Operator 统一管理并部署在每个工作节点上，用户的作业运行时将使用节点上已部署的 MXMACA® SDK。此过程在操作层面对用户透明，尽管用户能够感知到容器在运行时自动安装了 MXMACA® SDK 这一变化，但无需为此做任何额外操作。

- **驱动自动部署**

  使用 GPU Extensions 方案时，系统管理员需预先在每个工作节点上安装内核态驱动（包含虚拟化驱动的配置及安装）。

  GPU Operator 方案提供了自动部署驱动及配置 GPU 虚拟化规格的能力。管理员可配置 `driver.deployPolicy` 选取不同部署策略或关闭内核态驱动的自动部署功能。

[1] kube-apiserver 及 kubelet 应保持配置 `--allow-privileged=true`

GPU Operator 提供名为 `driver-config` 的 `ConfigMap`，管理员可通过此文件指定工作节点的 GPU 虚拟化规格（vfnum），配置文件的更新将在几分钟内在相关节点上生效。

- **方案自动化部署**

  GPU Operator 方案提供的云原生能力由多个软件组件协作完成，组件部署时存在参数配合，部署时序也有严格的要求，安装这些组件是一件极其复杂的事情。

  因此，正如 GPU Operator 字面所传达出来的信息，此方案基于 Operator Framework[2] 实现，组件部署及同步全部由代码控制、自动化地进行。管理员通过一条 `helm install` 命令即可完成部署，而无需关注背后的细节。

3.2 部署参考

3.2.1 安装 GPU Operator

在 Kubernetes 管理节点执行如下命令，即可完成 GPU Operator 的安装：

```
helm install oci://DOMAIN/PROJECT/metax-operator \
  --create-namespace -n metax-operator \
  --generate-name \
  --wait \
  --set registry=DOMAIN/PROJECT
```

3.2.2 设置 Chart 选项

GPU Operator Chart 支持 GPU Extensions Chart 全部选项。额外支持的选项，参见下方表格。

表 3.1 Global Options

选项	类型	描述
gpuDevice	struct	gpu-device 组件选项，参考gpu-device Options
gpuLabel	struct	gpu-label 组件选项，参考gpu-label Options
driver	struct	driver-manager 组件内核驱动控制选项, 参考Kernel Module Control Options
maca	struct	driver-manager 组件 MXMACA® 控制选项, 参考MXMACA Control Options
runtime	struct	container-runtime 组件选项，参考Container Runtime Options

表 3.2 Kernel Module Control Options

选项	类型	描述
deployPolicy	enum	资源部署策略，默认为 `IfNotPresent`，可选 `Always` 和 `Never`
payload	struct	内核驱动控制资源包选项，参考Driver Payload Options
log	struct	日志选项，参考Log Options

表 3.3 MXMACA Control Options

选项	类型	描述
payload	struct	MXMACA® 控制资源包选项，参考MXMACA Payload Options
log	struct	日志选项，参考Log Options

[2] https://operatorframework.io/

表 3.4 Container Runtime Options

选项	类型	描述
log	struct	日志选项，参考Log Options

表 3.5 Driver Payload Options

选项	类型	描述
name	string	payload 镜像名，默认值为 `driver-image`
version	string	payload 镜像版本，若 `driver.deployPolicy` 的值不为 `Never`，需显式指定
registry	string	payload 镜像地址，默认使用全局选项 `registry` 的值

表 3.6 MXMACA Payload Options

选项	类型	描述
images	string list	payload 镜像列表，格式为 `{IMAGE_1, IMAGE2, ...}`，其中 `IMAGE` 应使用完整格式显式指定 tag，如 `maca-c500:2.18.0.3-ubuntu18.04-amd64`
registry	string	payload 镜像地址，默认使用全局选项 `registry` 的值

3.2.3 配置虚拟化

GPU Operator 在将所有软件组件部署到集群上后，会生成一个名为 `driver-config` 的 `ConfigMap`。

```
{
  "node-vfnums": "nodes:
    sample-node:
      vfnum: 4
    sample-node2:
      vfnum: 2
    foo:
      vfnum: 2
  "
}
```

用户可通过修改其内容控制集群上节点的虚拟化设置，如新增 [line 7-8] 所示内容，将 foo 节点上的每张显卡分为 2 个 VF 实例。根据 `ConfigMap` 传递到工作节点所需的时间不同，管理员可在几分钟内观测到 foo 节点上 `metax-tech.com/gpu` 资源总数变为 $N_{(gpu)} \times M_{(vfnum)}$。期间，foo 节点上以 `metax-gpu-device` 开头命名的 `pod` 被自动销毁重建属于预期行为。

3.2.4 验证部署

GPU Operator 引入了更多的软件组件，并且具有更为复杂的组件间依赖关系，但验证部署的方式和 GPU Extensions 方案保持一致。无论使用何种部署方案，均需要在所有组件正常工作的前提下，节点上才有可分配的 `metax-tech.com/gpu` 资源，以此保证用户作业的安全。

详细信息，参见 GPU Extensions 的验证部署 章节。

3.2.5 卸载 GPU Operator

若需卸载 GPU Operator，可先通过 `helm list -n metax-operator` 命令查询已安装的 Chart，再执行 `helm uninstall <CHART> -n metax-operator --wait` 命令进行卸载。

注解： -n 指定的参数为 namespace，应与安装时所指定的值保持一致。

也可直接运行以下脚本自动检查已安装的 GPU Operator 并进行卸载。

```
chart=$(helm list -q -f "metax" -n metax-operator)
if [[ -n $chart ]]; then
    helm uninstall $chart -n metax-operator --wait
fi
```

注解： 为确保 GPU Operator 部署的资源能够被正确清理，--wait 参数为必选项，不可省略。

3.3 提交作业

3.3.1 制作容器镜像

1. **加载基础镜像**

   首先需准备镜像，参考MXMACA® 容器镜像 章节加载镜像。

2. **准备 Dockerfile**

   在 GPU Operator 方案部署的集群中，应用容器镜像无需携带 MXMACA® 软件栈，因此尺寸更加小巧，同时在有必要升级 MXMACA® 版本时，应用容器镜像也无需重建。尽管应用容器镜像可以不包含 MXMACA® 的文件，但容器构建过程中仍然需要 MXMACA® 的基础镜像作为构建环境。推荐在 Dockerfile 使用 multi-stage build 功能。

   以下为 Dockerfile 示例：

```
FROM cr.metax-tech.com/library/maca-c500:2.18.0.3-ubuntu18.04-amd64 as builder

RUN apt-get update && apt-get install -y \
  autoconf \
  build-essential \
  libtool \
  patchelf \
  pkg-config \
  ...

ENV VAR1=VALUE1
ENV VAR2=VALUE2
...

WORKDIR /workspace
COPY . /workspace
RUN make OUTPUT=/app

FROM ubuntu:18.04

RUN apt-get update && apt-get install -y \
  libelf1  \
  libltdl7 \
  libnuma1 \
```

(下页继续)

(续上页)

```
25    ... \
26 && rm -rf /var/lib/apt/lists/*
27
28 COPY --from=builder /app /app
```

用户基于此示例编写适用于实际工程的 Dockerfile 时，需注意以下细节：

- [line 1] 使用 MXMACA® 基础镜像作为 `builder` 镜像，提供必要的 MXMACA® 软件栈工具及库文件。
- [line 9] 此处增加构建用户应用所必须的工具和开发依赖库。
- [line 13] 此处设置构建过程所需的环境变量。
- [line 19] 基于 `ubuntu:18.04` 镜像而非 MXMACA® 基础镜像。
- [line 25] 修改此处以安装用户应用运行时依赖库，`libelf1`、`libltdl7` 和 `libnuma1` 为 MXMACA® SDK 运行时依赖。
- [line 28] 从 `builder` 容器中拷贝构建好的应用至基础容器。

使用此过程构建应用容器可避免因开发工具、开发依赖库以及构建过程临时文件造成的存储浪费。

3. **构建镜像**

假设用户工作目录有如下结构，Project 为应用项目根目录，Dockerfile 放置于同级目录。

```
workspace
├── Dockerfile
└── Project
    ├── Makefile
    └── src
```

用户可在 workspace 目录运行如下命令完成镜像的构建：

```
docker build -f Dockerfile -t DOMAIN/PROJECT/application:1.0 ./Project
```

3.3.2 准备作业 yaml 文件

用户可参考如下示例编写作业 yaml 文件：

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: application
5    labels:
6      app: application
7  spec:
8    nodeSelector:
9      metax-tech.com/gpu.vfnum: "0"
10     metax-tech.com/gpu.product: "C500"
11   containers:
12   - image: DOMAIN/PROJECT/application:1.0
13     name: application-container
14     command: ["/bin/bash", "-c", "--"]
15     args: ["/app/bin/xxx --option=val; sleep infinity"]
16     resources:
17       limits:
18         metax-tech.com/gpu: 2
```

3.3.3 提交作业

执行 `kubectl create -f sample.yaml` 命令提交作业。

4 MetaX Docker

曦云系列 GPU 提供的 metax-docker 工具，是基于官方 Docker 的扩展工具，用于解决 GPU 选择过程繁琐以及应用镜像体积庞大的问题。用户可独立发布不包含 MXMACA® 软件栈的容器镜像，并借助 metax-docker 工具在运行时获得一个完整的运行环境。

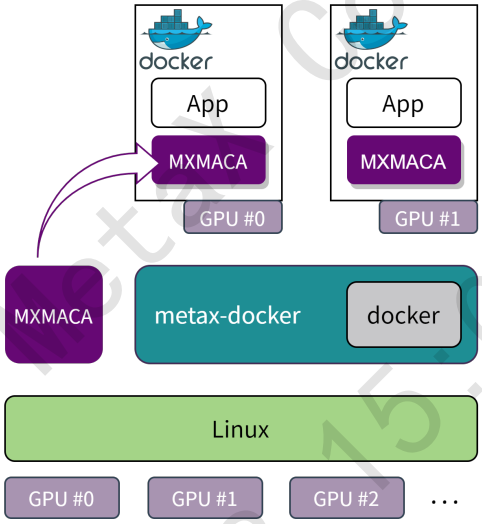

图 4.1 MetaX Docker

4.1 安装 metax-docker

安装 metax-docker 需要使用名称格式为 `metax-docker_0.8.0.tar` 的离线压缩包。用户可解压后根据安装环境选择合适的包进行安装。

操作步骤

1. 在终端上使用以下命令安装 metax-docker。

```
mkdir metax-docker
tar -C metax-docker -xvzf metax-docker_0.8.0.tar
cd metax-docker
sudo ./metax-docker_0.8.0.<ARCH>.run
```

4.2 使用 metax-docker

用户需要安装版本 ≥19.03 的 Docker 工具。同时应确保主机上已经正确安装了 MXMACA® 软件栈。

用户可通过执行如下命令在容器中使用曦云 GPU：

```
metax-docker run -it --rm --gpus=all user-application:1.0 /bin/bash
```

metax-docker 支持官方 Docker 的全部命令及参数，并在 `run` 命令下支持额外参数，具体参见下表。

表 4.1 metax-docker 额外支持的参数

参数	说明
–gpus	all：将全部卡传递至容器 N：将枚举的前 N 张卡传递至容器 [ID1,ID2…]：将 ID 值为 ID1 和 ID2 的卡传递至容器 [SN1,SN2…]：将 SN 值为 SN1 和 SN2 的卡传递至容器
–maca	若希望使用默认安装路径之外的 MXMACA®，可使用此参数指定
–maca-force	默认情况下，若容器内/opt/maca 路径下已存在 MXMACA®，metax-docker 会停止将主机上的 MXMACA® 安装进容器。设置此参数可强制容器使用主机上的 MXMACA®。

4.3 构建应用软件镜像

用户可根据此节内容构建更加适用于 MetaX Docker 以及云场景的容器镜像。

操作步骤

1. 假设用户软件具有如下目录结构，源代码放置在项目的 src 目录下。

```
samples/
  ├──Dockerfile
  └──src
     ├──Makefile
     ├──sample1.cpp
     └──sample2.cpp
```

2. 用户可使用如下内容的 Dockerfile 制作应用软件镜像。

```
#intermediate builder image
FROM cr.metax-tech.com/library/mxc500-maca:2.24.0.0 AS builder

WORKDIR/workspace

COPY src src

#build user application and put artifacts under /workspace/dist

RUN make -C ./src OUT=./dist

#artifact mage
FROM ubuntu:20.04
RUN apt-get update && \
  apt-get install -y libnuma1 libelf1 && \
  apt-get clean
COPY --from=builder /workspace/dist /applications
CMD ["/applications/foo"]
```

3. 完成 Dockerfile 的编写后，用户可在项目根目录下执行如下命令完成镜像的构建。

```
docker build -t user-application:1.0 .
```