XPU 安装及使用手册

文档版本号：2.0.23

发布日期：2023-02

声明

YOYOWORKS、优优、优优工场、XPU 和其徽标是北京优优工场科技有限公司在中国和/或其他国家/地区的注册商标或商标。其他公司名称或产品名称仅作提供信息之用，可能是其各自所有者的商标。

注意

您购买的产品、服务或特性等应受优优公司商业合同和条款的约束，本文档中描述的全部或部分产品、功能或特性可能不在您购买或使用的范围之内。除非合同另有约定，优优公司对本文档内容不做声明或保证。

由于产品版本升级或其他原因，本文档会不定期进行更新内容。除非另有约定，本文档仅作为使用指导，本文档中的所有描述不构成任何担保。

目录

1 XPU 简介

XPU 是北京优优工场科技有限公司（YOYOWORKS，以下简称“优优”）推出的 Linux 容器环境下 GPU 虚拟化产品。XPU 的核心思想是将 GPU 在 Linux 内核层进行切分，向上系统和应用模拟出统一的 XPU 设备供容器使用，实现多个容器独立安全地共享使用一张 GPU 卡的资源（也可以共享使用多张 GPU 卡）。XPU 提供了一套框架很好地解耦了 AI 等 GPU 应用（TensorFlow，PyTorch 等）与 GPU 物理卡之间的强绑定关系，实现了容器间 GPU 应用在同时使用 GPU 资源时的算力、显存和故障隔离，从而在保证 AI 应用可用性和安全性的前提下，显著地提高了 GPU 硬件资源的使用效率，大大提供了 GPU 集群的调度和管理能力。

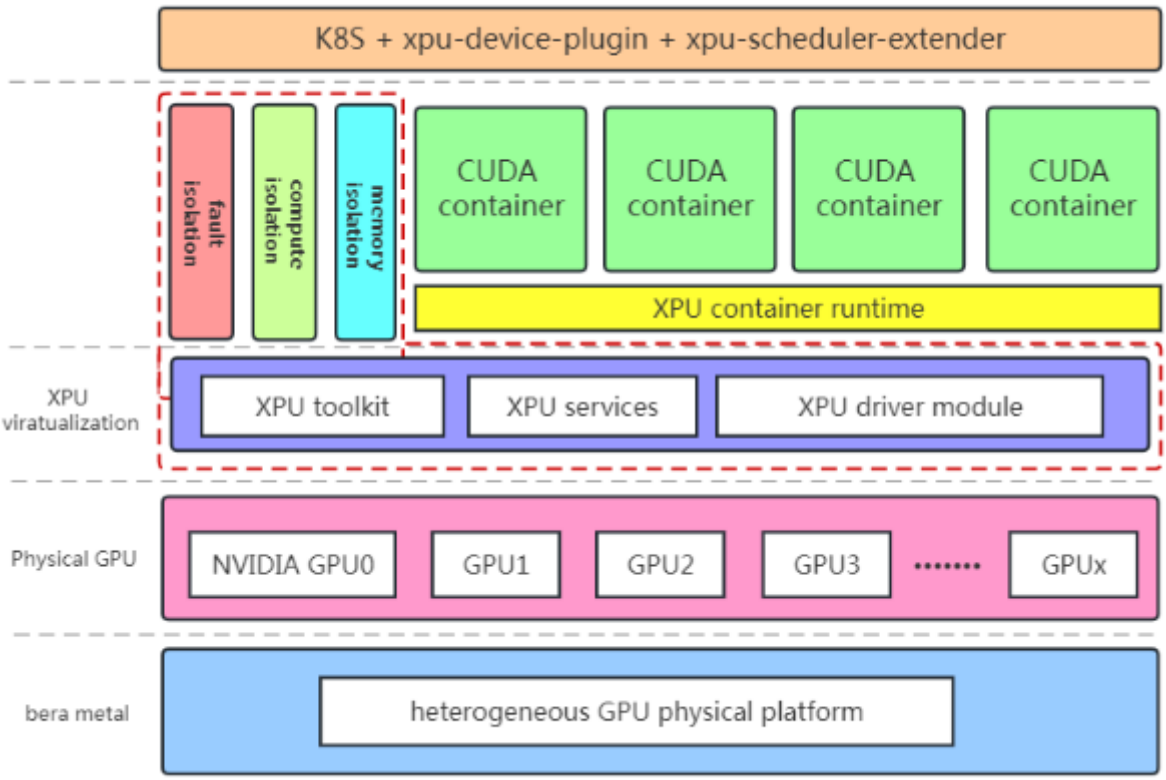

图 1 XPU 架构示意图

XPU 的实现主要包括两大部分 系统部分及容器部分。系统部分包括 XPU service 以及 XPU modules，其中运行在 Linux Kernel 内核态的 XPU modules 把单个 GPU 卡动态划分为多个兼容的 XPU 卡分配给容器使用和访问，并且在主机操作系统内核层面实现多个容器的 GPU（XPU）算力和显存资源的 QoS 控制和管理，在确保容器间安全隔离的前提下，实现 GPU 卡在多个容器间的高效共享使用。容器部分为 XPU container runtime，作为 Docker container 的对应工具，主要实现将 XPU 设备与容器间的映射、分配和回收管理，让容器内的应用能够识别虚拟的 GPU 设备(XPU)，从而实现透明地运行原容器内的 GPU 相关应用负载。

2 环境准备

软硬件要求

硬件要求	软件要求
● CPU： Intel/AMD 64 位 ● 内存： 16GB（最少） ● GPU： NVIDIA GPU ● 网络： Internet 在线	● OS：CentOS 7.x/8.x (64bit) RHEL7.x/8.x/9.x (64bit) AlmaLinux 8.x/9.x (64bit) Rocky Linux 8.x/9.x (64bit) Ubuntu 18.04/20.04/22.04 LTS (64bit) ● Docker 已安装（>= 19.03） ● NVIDIA GPU 驱动已安装（>= 440.x.x）
注： 1. XPU 只支持 Pascal 及以后架构的 GPU（包括 Tesla/Quadro/GeForce 全系列）	注： 1. XPU 只支持 64 位 OS 2. XPU 只支持 64 位应用

首先基于相应的操作系统安装 Docker

1. Ubuntu 系统安装示例

```
#>sudo apt-get install -y docker
#>sudo apt-get install -y docker.io
#>sudo apt-get install -y docker-registry
```

2. RHEL/CentOS 系统安装示例

RHEL/CentOS7.x

```
#>sudo yum install -y yum-utils device-mapper-persistent-data lvm2
#>sudo yum-config-manager --add-repo https://download.docker.com/linux/CentOS/docker-ce.repo
#>sudo yum install -y docker-ce
#>sudo systemctl enable --now docker
```

RHEL/CentOS8.x

```
#>sudo dnf config-manager --add-repo=https://download.docker.com/linux/CentOS/docker-ce.repo
#>sudo dnf remove runc podman buildah
#>sudo dnf install -y docker-ce
#>sudo systemctl enable --now docker
```

3. 检查安装结果，运行 docker version，出现以下提示表示安装成功

```
ubuntu@ubuntu198:~$ sudo docker version
Client: Docker Engine - Community
 Version:           20.10.8
 API version:       1.41
 Go version:        go1.16.6
 Git commit:        3967b7d
 Built:             Fri Jul 30 19:54:08 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.8
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.6
  Git commit:       75249d8
  Built:            Fri Jul 30 19:52:16 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
 nvidia:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
ubuntu@ubuntu198:~$
```

在系统上安装 NVIDIA GPU 驱动程序

1. 选择对应系统版本的 NVIDIA GPU 驱动并下载：

https://www.nvidia.com/Download/index.aspx

2. 安装（以 Ubuntu20.04 和 CentOS8.4 为例）：

Ubuntu20.04

```
#>sudo cat >>/etc/modprobe.d/blacklist-nouveau.conf<<EOF
blacklist nouveau
options nouveau modeset=0
EOF
#>sudo update-initramfs -u
#>sudo reboot
#>sudo apt-get install -y build-essential gcc-multilib
#>sudo sh./NVIDIA-Linux-x86_64-510.47.03.run -s
```

CentOS8.4 安装 kernel-devel 时请注意要匹配当前 kernel 版本（uname -r 查看)

```
#>sudo cat >>/etc/modprobe.d/blacklist.conf<<EOF
blacklist nouveau
options nouveau modeset=0
EOF
#>sudo dracut --force
#>sudo reboot
#>sudo yum update
#>sudo yum install epel-release libstdc++.i686 kernel-devel
#>sudo sh./NVIDIA-Linux-x86_64-510.47.03 -s
```

3. 检查结果，运行 nvidia-smi，如果出现以下画面，表示安装成功：

```
ubuntu@ubuntu198:~$ sudo nvidia-smi
Fri Sep 10 03:23:15 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.57.02    Driver Version: 470.57.02    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 A10          On   | 00000000:25:00.0 Off |                    Off |
|  0%   41C    P8    24W / 150W |      0MiB / 24258MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A10          On   | 00000000:26:00.0 Off |                    Off |
|  0%   41C    P8    24W / 150W |      0MiB / 24258MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A10          On   | 00000000:29:00.0 Off |                    Off |
|  0%   38C    P8    22W / 150W |      0MiB / 24258MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A10          On   | 00000000:2D:00.0 Off |                    Off |
|  0%   39C    P8    24W / 150W |      0MiB / 24258MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
ubuntu@ubuntu198:~$
```

在系统上安装 NVIDIA Container Toolkit

请参考官方文档进行安装。

到此，环境准备工作完成，接下来安装 XPU 相关软件包。

3 安装 XPU

从以下链接下载 XPU 安装脚本到需要安装 XPU 的机器上：

```
#>sudo curl http://www.openxpu.com/release/xpu-installer.sh -o xpu-installer.sh
```

运行该脚本：

```
#>sudo chmod +x xpu-installer.sh
#>sudo ./xpu-installer.sh
```

该脚本会根据当前系统 OS 类型，版本，对环境进行检测，如果系统符合 XPU 安装条件，下载对应的 rpm 或者 deb 包（注：具体的版本可能因您安装的具体版本而有差别）

Ubuntu 系统包含以下一些包：

```
xpu-container_2.0.22-6_amd64.deb
xpu-modules_2.0.22-6*_amd64.deb
xpu-service_2.0.22-6_amd64.deb
xpu-agent_2.0.22-6_amd64.deb
```

CentOS 系统包含以下一些包：

```
xpu-container-2.0.22-6.x86_64.rpm
xpu-modules-2.0.22-6*.x86_64.rpm
xpu-service-2.0.22-6.x86_64.rpm
xpu-agent-2.0.22-6.x86_64.rpm
```

检查 XPU 相关软件是否已经正确安装并运行：

```
#>sudo systemctl status xpu
```

```
ubuntu@ubuntu198:~$ sudo systemctl status xpu
● xpu.service - XPU Control and Monitor Daemon
   Loaded: loaded (/lib/systemd/system/xpu.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-09-09 10:32:17 UTC; 17h ago
     Docs: https://xpu.yoyoworks.com
 Main PID: 16911 (xpud)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/xpu.service
           └─16911 /usr/bin/xpud --log=error

Sep 09 10:32:17 ubuntu198 systemd[1]: Started XPU Control and Monitor Daemon.
ubuntu@ubuntu198:~$
```

```
#>sudo systemctl status xpuagent
```

```
ubuntu@ubuntu:~$ sudo systemctl status xpuagent
• xpuagent.service - XPU RESTful API access Daemon
     Loaded: loaded (/lib/systemd/system/xpuagent.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-07-12 03:12:22 UTC; 3 days ago
       Docs: https://xpu.yoyoworks.com
   Main PID: 1963 (xpuagent)
      Tasks: 4 (limit: 18998)
     Memory: 19.3M
     CGroup: /system.slice/xpuagent.service
             └─1963 /usr/bin/xpuagent --log=error

Jul 12 03:12:22 ubuntu systemd[1]: Started XPU RESTful API access Daemon.
Jul 12 03:12:33 ubuntu xpuagent[1963]: xpu_agent_init succeeds.
ubuntu@ubuntu:~$
```

#>sudo lsmod |grep xpu

```
ubuntu@ubuntu:~$ lsmod|grep xpu
xpum                  303104  4
ubuntu@ubuntu:~$
```

4 授权 XPU

XPU 初次安装后，自带 Express 版本授权，如下图所示：

```
root@ubuntu:~# cat /proc/xpu/license
XPU Licensed by YOYOWORKS: Express
root@ubuntu:~#
```

Express 版本最多支持一台独立的服务器节上激活每块物理 GPU（TESLA/QUADRO）上 2 个 XPU 的能力（仍然可以传统方式使用其他 GPU 卡），GEFORCE 卡不受此限制。如果需要激活一台独立节点上更多 GPU 卡的 XPU 虚拟化功能，请首先使用 XPU 软件包自带的 xpu-gpu-collect 工具收集 GPU 信息，并将该信息发送至 support@yoyoworks.com 获取软件授权和支持。

如果已从优优获取到 XPU 授权文件（一般情况下为 company_name.asc），请将该授权放到 GPU 服务器节点的/etc/xpu/license/目录下，并重启 XPU 服务使授权立即生效：

```
ubuntu@ubuntu:~$ sudo systemctl restart xpu
ubuntu@ubuntu:~$ sudo systemctl status xpu
● xpu.service - XPU Control and Monitor Daemon
     Loaded: loaded (/lib/systemd/system/xpu.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-02-21 04:37:51 UTC; 10s ago
       Docs: https://xpu.yoyoworks.com
   Main PID: 22882 (xpud)
      Tasks: 1 (limit: 19004)
     Memory: 5.9M
     CGroup: /system.slice/xpu.service
             └─22882 /usr/bin/xpud --log=error

Feb 21 04:37:51 ubuntu systemd[1]: Started XPU Control and Monitor Daemon.
```

查看 XPU 当前的授权信息，根据不同的授权会显示相应的授权信息：

```
root@ubuntu:~# cat /proc/xpu/license
XPU Licensed by YOYOWORKS: Advanced
root@ubuntu:~#
```

如果后续有该服务器节点添加了新的 GPU 卡（或更换了 GPU），或者授权信息有变化，将新的授权文件放在/etc/xpu/license/下，重启 XPU 服务即可更新授权信息。

XPU 版本划分

XPU 缺省安装后自动获得 Express 版本 如果需要其他版本的 XPU 请联系 support@yoyoworks.com 获取软件授权和支持。

版本	功能 & 限制
Express	TESLA/QUADRO 型号显卡限制每块物理显卡创建最多 2 个 XPU 该版本不支持算力控制
Advanced	支持所有类型显卡全功能

5 使用 XPU

容器使用虚拟 GPU（XPU）的基本概念

XPU 将物理 GPU 切分成不同的份额来供容器使用，我们称之为 shares，shares 由显存和算力共同组成。运行 XPU 容器时需要将不同 GPU 的 shares 通过容器的环境变量配置给容器使用。XPU 会根据 shares 环境变量为容器准备对应的 GPU 资源，分配相应的 shares。XPU 通过切分、控制和管理不同容器的 shares，来实现 GPU 在多个容器间共享，并且通过运行在 Linux kernel 内核态的 XPU modules 内核模块全生命周期内保证不同容器间使用虚拟 GPU（XPU）的显存、算力及故障的完全隔离。

配置容器使用虚拟 GPU（XPU）

如下面的示例所示，配置：

```
#>sudo docker run -itd --gpus all --runtime=nvidia --name xpu -v /mnt:/mnt -e OPENXPU_XPU_SHARES=0:4096-50% nvcr.io/nvidia/cuda:11.5.0-devel-centos7 /bin/bash
```

docker 环境变量也可以使用 GPU UUID 表示使用的 GPU：

OPENXPU_XPU_SHARES=GPU-c5963e55-4cc8-359b-d2d8-b8d4ba5bd92d:4096-50%

显存也可以用百分比设置：

OPENXPU_XPU_SHARES=0:50%-50%

环境变量	取值类型	说明	示例
OPENXPU_XPU_SHARES	string	当前容器使用的 XPU shares 数（shares 数包含的显存和算力可独立控制，显存必须配置，算力不配置表示不控制）及 GPU	在 1 台有 4 张显卡的机器上，执行 nvidia-smi -L 查看 GPU 显卡设备号及 UUID。 返回示例如下所示： GPU 0: Tesla T4 (UUID: GPU-3aec****) GPU 1: Tesla T4 (UUID: GPU-45bc****) GPU 2: Tesla T4 (UUID: GPU-e728****) GPU 3: Tesla T4 (UUID: GPU-403e****)

		分配情况	设置以下环境变量： **OPENXPU_XPU_SHARES=0:4096-50%,2:4096-50%H** 该写法表示为该容器分配第 0 张显卡，shares 为显存 4096MB，算力 50%，非高优先级模式；分配第 2 张显卡，shares 为显存 4096MB，算力 50%，于高优先级模式；

注：

H 表示高优先级模式，属于绝对分配。能保证容器相应显存和算力的使用，XPU 的显存分配和算力分配不能超过物理 GPU 限制；缺省（不带 H）表示资源共享，根据当时 GPU 资源使用情况进行相对分配。

OPENXPU_XPU_SHARES=0:4096（或：OPENXPU_XPU_SHARES=0:50%）中算力无配置，则表示算力不受控制；

OPENXPU_XPU_SHARES=0:4096-50%（或：OPENXPU_XPU_SHARES=0:50%-50%），算力分配 50%；

OPENXPU_XPU_SHARES=0:4096-50%H（或：OPENXPU_XPU_SHARES=0:50%-50%H）表示资源高优先级，显存保证 4096MB，算力保证最低 50%，同一 GPU 上高优先级的算力总和不能超过 100%；

在同一个 GPU 上不能设置算力不受控和算力控制两种模式，即：

A 容器配置 OPENXPU_XPU_SHARES=0:4096（或：OPENXPU_XPU_SHARES=0:50%），B 容器配置 OPENXPU_XPU_SHARES=0:4096-50%(或 :OPENXPU_XPU_SHARES=0:50%-50%)时 ,不能将 A、B 容器同时应用在同一 GPU 上，即不同容器的算力策略需要兼容。

查看某 GPU 卡显存及算力分配信息：

```
#>sudo cat /proc/xpu/nvidia0/all_memory
#>sudo cat /proc/xpu/nvidia0/free_memory
#>sudo cat /proc/xpu/nvidia0/weights
```

查看 nvidia0 设备的 all_memory/free_memory，表示该设备可以显存总数及分配情况，该目录下的 weights 表示算力分配情况。

登录对应容器查看：

```
#>sudo docker exec -it xpu /bin/bash
```

```
# container>nvidia-smi
```

```
ubuntu@ubuntu198:~$ sudo nvidia-smi
Fri Sep 10 03:23:15 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.57.02    Driver Version: 470.57.02    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 A10          On   | 00000000:25:00.0 Off |                    Off |
|  0%   41C    P8    24W / 150W |      0MiB / 12128MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A10          On   | 00000000:26:00.0 Off |                    Off |
|  0%   41C    P8    24W / 150W |      0MiB / 12128MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A10          On   | 00000000:29:00.0 Off |                    Off |
|  0%   38C    P8    22W / 150W |      0MiB / 12128MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A10          On   | 00000000:2D:00.0 Off |                    Off |
|  0%   39C    P8    24W / 150W |      0MiB / 12128MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
ubuntu@ubuntu198:~$
```

在 Kubernetes 集群中使用 XPU

在 K8S 集群中使用 XPU，主要利用了 K8S 1.8 版本后提出的 Extended Resources 和 Device Plugin 方案。Device Plugin：K8S 制定设备插件接口规范，定义异构资源的上报和分配，设备厂商只需要实现相应的 API 接口，无需修改 kubelet 源码即可实现对其他硬件设备的支持。Extended Resource（XPU 定义的 extended resource 为 openxpu.com/xpu-shares），K8S scheduler 可以根据 Pod 的创建删除计算资源可用量，而不再局限于 CPU 和内存的资源统计，进而将有特殊资源需求的 Pod 调度到相应的节点上。

在 K8S 的使用过程中，除了安装前文所述 XPU driver module 和 XPU container runtime 之外，需要用到 xpu-device-plugin 和 xpu-extend-scheduler 两个插件，具体使用如下：

1. 确保 Kubernetes 集群已经正确安装，版本 >= 1.18；

2. 存在 GPU 的 node 上打上 xpu=true 的标签：

```
#>sudo kubectl label node <node_name> xpu=true
```

查询如下图所示：

```
ubuntu@ubuntu198:~$ sudo kubectl get node ubuntu198 --show-labels
NAME        STATUS     ROLES    AGE     VERSION   LABELS
ubuntu198   NotReady   <none>   3m49s   v1.19.4   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/c
8,kubernetes.io/os=linux,xpu=true
ubuntu@ubuntu198:~$
```

3. 部署 xpu-device-plugin 插件：

```
#>sudo kubectl apply -f http://www.openxpu.com/release/latest/k8s-plugin/device-plugin-rbac.yaml
#>sudo kubectl apply -f http://www.openxpu.com/release/latest/k8s-plugin/device-plugin-ds.yaml
```

4. 部署 xpu-extend-scheduler 插件：

```
#>sudo wget http://openxpu.com/release/latest/k8s-plugin/scheduler-policy-config.json
```

修改 K8S scheduler 的配置文件，/etc/kubernetes/manifests/kube-scheduler.yaml，增加：

```
- --policy-config-file=/etc/kubernetes/scheduler-policy-config.json
- --use-legacy-policy-config=true


- mountPath: /etc/kubernetes/scheduler-policy-config.json
name: scheduler-policy-config
readOnly: true


- hostPath:
path: /etc/kubernetes/scheduler-policy-config.json
type: FileOrCreate
name: scheduler-policy-config
```

应用 yaml 文件：

```
#>sudo kubectl apply -f http://www.openxpu.com/release/latest/k8s-plugin/xpu-scheduler-extender.yaml
```

5. 检查 xpu-device-plugin 和 xpu-extend-scheduler 两个插件是否正常运行：

```
ubuntu@ubuntu198:~$ sudo kubectl get po --all-namespaces
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-system   coredns-6d56c8448f-lfx7h              1/1     Running   0          25m
kube-system   coredns-6d56c8448f-xk5d8              1/1     Running   0          25m
kube-system   etcd-ubuntu202                        1/1     Running   0          25m
kube-system   kube-apiserver-ubuntu202              1/1     Running   0          25m
kube-system   kube-controller-manager-ubuntu202     1/1     Running   0          25m
kube-system   kube-flannel-ds-8lc78                 1/1     Running   0          20m
kube-system   kube-flannel-ds-hfdx6                 1/1     Running   0          20m
kube-system   kube-proxy-6h2ql                      1/1     Running   0          21m
kube-system   kube-proxy-r9cc5                      1/1     Running   0          25m
kube-system   kube-scheduler-ubuntu202              1/1     Running   0          15m
kube-system   xpu-device-plugin-ds-jk8mx            1/1     Running   0          72s
kube-system   xpu-scheduler-extender-dd9968644-brmnv  1/1     Running   0          10m
ubuntu@ubuntu198:~$
```

6. 测试，将 XPU 定义的 extended resource 写入需要调度的 Pod yaml 文件中，其中 xpu-shares 代表了需要申请的显存大小（单位：1GB），xpu-device-plugin 在分配算力时根据设置的显存所占 GPU

显存比例自动设置，通过 K8S 调度的 xpu-shares 设置只能在单张显卡上，即多 GPU 主机会将最合适分配的那张 GPU 提供给 Pod 使用。

注意：所有通过 K8S 调度的资源均为可共享资源。

```
spec:
  template:
    spec:
      containers:
      - name: xpu-sam
        securityContext:
          privileged: true
          capabilities:
            add: ["SYS_PTRACE"]
        command: ["/bin/bash", "-c", "cd /data/tensorflow/alexnet; python alexnet_benchmark.py"]
        env:
        - name: PATH
          value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        image: registry.cn-beijing.aliyuncs.com/yoyoworks/tf-example:v1.0
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: "4"
            memory: 8Gi
            openxpu.com/xpu-shares: 2
          requests:
            cpu: "4"
            memory: 8Gi
            openxpu.com/xpu-shares: 2
      restartPolicy: Never
```

或者直接从优优网站下载一个示例 yaml 文件直接应用：

```
#>sudo kubectl apply -f http://www.openxpu.com/release/latest/k8s-plugin/xpu-sam.yaml
```

6 测试 XPU

下载一个测试脚本并运行，该脚本会自动将测试用例及环境准备好：

```
#>sudo curl http://www.openxpu.com/support/test/xpu-test.sh -o xpu-test.sh
#>sudo chmod +x ./xpu-test.sh
#>sudo ./xpu-test.sh -i 1 -v 11.2.0 -m 2048
```

查看容器是否正常启动：

```
ubuntu@ubuntu198:~$ sudo docker ps -a --format "table {{.ID}}\t{{.Names}}\t{{.Status}}\t{{.Ports}}"
CONTAINER ID   NAMES     STATUS         PORTS
d809c96f3d94   tf0-1-1   Up 15 hours    6006/tcp, 8888/tcp
ubuntu@ubuntu198:~$
```

检查 tensorflow 任务是否使用了 gpu，使用命令 nvidia-smi pmon -d 1：

```
ubuntu@ubuntu198:~$ nvidia-smi pmon -d 1
# gpu        pid  type    sm   mem   enc   dec   command
# Idx          #   C/G     %     %     %     %   name
    0      17216     C    49    39     -     -   python
    0      17216     C    49    39     -     -   python
    0      17216     C    51    41     -     -   python
    0      17216     C    52    41     -     -   python
    0      17216     C    50    41     -     -   python
    0      17216     C    51    41     -     -   python
    0      17216     C    49    40     -     -   python
    0      17216     C    48    39     -     -   python
    0      17216     C    50    40     -     -   python
    0      17216     C    52    41     -     -   python
```