docker使用GPU总结



在docker容器中使用显卡

## 一 docker19.03以前的事情

### 1.1 指定显卡硬件名

最初的容器中使用显卡，需要指定硬件名。经历了两种方式

1. 使用lxc驱动程序运行docker守护进程，以便能够修改配置并让容器访问显卡设备（非常麻烦，参考链接中最久远的回答）

2. Docker 0.9中放弃了lxc作为默认执行上下文，但是依然需要指定显卡的名字

（1）找到你的显卡设备

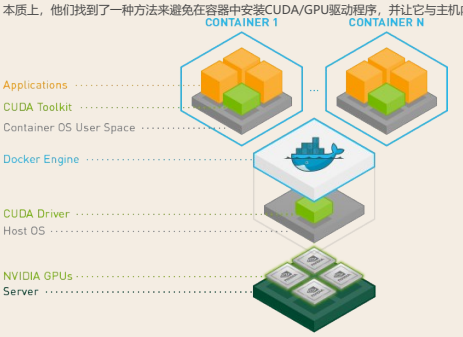
```
ls -la /dev | grep nvidia

crw-rw-rw-  1 root root    195,   0 Oct 25 19:37 nvidia0
crw-rw-rw-  1 root root    195, 255 Oct 25 19:37 nvidiactl
crw-rw-rw-  1 root root    251,   0 Oct 25 19:37 nvidia-uvm
```

（2）启动容器时，指定显卡设备

```
sudo docker run -ti --device /dev/nvidia0:/dev/nvidia0 --device /dev/nvidiactl:/dev/nvidiactl --device /dev/nvidia-uvm:/dev/nvidia-uvm tleyden5iwx/ubuntu-cuda /bin/bash
```

### 1.2 nvidia-docker

英伟达公司开发了nvidia-docker，该软件是对docker的包装，使得容器能够看到并使用宿主机的nvidia显卡.本质上，他们找到了一种方法来避免在容器中安装CUDA/GPU驱动程序，并让它与主机内核模块匹配。




测试：

```
# Install nvidia-docker and nvidia-docker-plugin
wget -P /tmp https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.1/nvidia-docker_1.0.1-1_amd64.deb
sudo dpkg -i /tmp/nvidia-docker*.deb && rm /tmp/nvidia-docker*.deb

# Test nvidia-smi
nvidia-docker run --rm nvidia/cuda nvidia-smi
```

指定使用两张卡：

```
docker run --rm --gpus 2 nvidia/cuda nvidia-smi
```

更详细得得用法见：User Guide — NVIDIA Cloud Native Technologies documentation

另外要注意nvidia-docker包括nvidia-docker1 和 nvidia-docker2，两者命令有一定差异

## 二 docker 19.03

感觉docker已是toB公司的必备吧，有了docker再也不用担心因客户环境问题导致程序各种bug，也大大省去了配置客户服务器的过程。

很多机器学习项目要使用GPU，所以需要docker支持GPU，在docker19以前的版本都需要单独下载nvidia-docker1或nvidia-docker2来启动容器，自从升级了docker19后跑需要gpu的docker只需要加个参数–gpus all即可(表示使用所有的gpu，如果要使用2个gpu：–gpus 2，也可直接指定哪几个卡：–gpus ‘“device=1,2”’，后面有详细介绍)。

接着需要安装nvidia驱动，这需要根据自己显卡安装相应的驱动，网上有很多类似教程，此处不再介绍。

不要以为这样就可以安心的使用gpu了，你的镜像还必须要有cuda才行，这也很简单，去dockerhub上找到和自己tensorflow相对应的cuda版本的镜像，再基于该镜像生成自己的镜像就可以轻松使用gpu了。这里需要额外多说一句，如果你的docker 本身就基于了某个镜像（例如基于本公司仓库的镜像），docker是不允许from两个镜像的，要想实现基于两个或多个，只能基于其中一个，其他的镜像通过各镜像的Dockerfile拼到新的Dockerfile上，但更多的镜像是没有Dockerfile的，可以通过docker history查看某镜像的生成过程，其实就是Dockerfile，nvidia/cuda官网本身就有Dockerfile，也可直接参考。

小结：要想在容器内使用gpu，每一层需要做好的事情

1、k8s：需要安装nvidia-device-plugin支持gpu调度。详见 https://www.cnblogs.com/linhaifeng/p/16111733.html

2、容器内：需要有cuda库

3、docker：安装docker19.03，低于此版本需要额外安装插件

4、操作系统：安装cuda驱动程序

5、硬件：插入gpu卡

### 2.1 安装toolkit

关于配置docker19使用gpu，其实只用装官方提供的toolkit即可，把github上的搬下来：
**Ubuntu 16.04/18.04, Debian Jessie/Stretch/Buster:**

```
# Add the package repositories
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee
/etc/apt/sources.list.d/nvidia-docker.list

$ sudo apt-get update && sudo apt-get install -y nvidia-container-toolkit
$ sudo systemctl restart docker
```

**CentOS 7 (docker-ce), RHEL 7.4/7.5 (docker-ce), Amazon Linux 1/2**

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.repo | sudo tee
/etc/yum.repos.d/nvidia-docker.repo

$ sudo yum install -y nvidia-container-toolkit
$ sudo systemctl restart docker
```

## 2.2 测试安装是否成功

**经过以上大部分linux系统的docker toolkit应该都能安装成功，如不能安装成功，可参考github官网，查看是否安装成功：**
**(1) 查看–gpus 参数是否安装成功：**

```
$ docker run --help | grep -i gpus
      --gpus gpu-request               GPU devices to add to the container ('all' to pass all GP
Us)
```

(2) 运行nvidia官网提供的镜像，并输入nvidia-smi命令，查看nvidia界面是否能够启动：

```
docker run --gpus all nvidia/cuda:9.0-base nvidia-smi
```

## 2.3 运行gpu的容器

从Docker 19.03开始，安装好docker之后，只需要使用 --gpus 即可指定容器使用显卡。

```
所有显卡都对容器可见：

docker run --gpus all --name 容器名 -d -t 镜像id

只有显卡1对容器可见：

docker run --gpus="1" --name 容器名 -d -t 镜像id

如果不指定 --gpus ，运行nvidia-smi 会提示Command not found
```

注意：

1. 显卡驱动在所有方式中，都要先安装好，容器是不会有显卡驱动的，一台物理机的显卡只对应一个显卡驱动，当显卡驱动安装好后（即使未安装cuda），也可以使用命令nvidia-smi

2. nvidia-smi显示的是显卡驱动对应的cuda版本，nvcc -V 显示的运行是cuda的版本
补充

```
启动容器时，容器如果想使用gpu，镜像里必须有cuda环境，就是说，针对想使用gpu的容器，镜像在制作时必须吧cuda环境打进去

 --gpus '"device=1,2"'，这个的意思是，将物理机的第二块、第三块gpu卡映射给容器？

下面三个参数代表的都是是容器内可以使用物理机的所有gpu卡
 --gpus all
 NVIDIA_VISIBLE_DEVICES=all
 --runtime=nvida

  NVIDIA_VISIBLE_DEVICES=2 只公开两个gpu，容器内只能用两个gpu
```

举例如下：

```
# 使用所有GPU
$ docker run --gpus all nvidia/cuda:9.0-base nvidia-smi

# 使用两个GPU
$ docker run --gpus 2 nvidia/cuda:9.0-base nvidia-smi

# 指定GPU运行
$ docker run --gpus '"device=1,2"' nvidia/cuda:9.0-base nvidia-smi
$ docker run --gpus '"device=UUID-ABCDEF,1"' nvidia/cuda:9.0-base nvidia-smi
```

ok，可以开心愉快的使用gpu了！！

后续

参考资料：

https://blog.csdn.net/qq_33547243/article/details/107433616

https://www.cnblogs.com/shoufu/p/12904832.html

https://github.com/NVIDIA/nvidia-docker/issues/533

首先介绍几个事实：

1. 最初的docker是不支持gpu的

2. 为了让docker支持nvidia显卡，英伟达公司开发了nvidia-docker。该软件是对docker的包装，使得容器能够看到并使用宿主机的nvidia显卡。

3. 根据网上的资料，从docker 19版本之后，nvidia-docker成为了过去式。不需要单独去下nvidia-docker这个独立的docker应用程序，只需要安装并配置一下nvidia-runtime，使用的时候用--gpus参数来控制。

(P.S.：因为本实验室服务器的docker默认是支持nvidia的runtime的，所以我在这里没有过多纠结，读者假如从零开始安装docker软件的话可能要细心地保证docker是支持gpu的docker)

然后我做了几个有代表性的实验：

1. docker run 的时候不加 --gpus参数，典型代码：

docker run -it --name test --rm ubuntu:latest

此时在容器内运行nvidia-smi会提示Command not found

2. docker run 的时候加上 --gpus参数，典型代码：

docker run -it --rm --name test --gpus all ubuntu:latest

此时在容器内运行nvidia-smi会有如下输出：

```
root@a9994eb6506c:/# nvidia-smi
Wed Feb 24 14:15:53 2021
+-----------------------------------------------------------------------------+
```

```
| NVIDIA-SMI 450.57       Driver Version: 450.57       CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| 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  Quadro RTX 6000     Off  | 00000000:5B:00.0 Off |                  Off |
| 33%   22C    P8    15W / 260W |   1624MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Quadro RTX 6000     Off  | 00000000:C8:00.0 Off |                  Off |
| 33%   23C    P8    16W / 260W |      3MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

从这两个实验我们可以得出结论，docker在启动容器的时候添加的--gpus参数确实是给容器添加了新东西的。比如/usr/bin/nvidia-smi这个可执行程序，如果你不添加--gpus参数是不会给你放到容器中的！此外可以推测，不加--gpus参数，宿主的gpu将对于容器不可见。

还有一个需要注意的点是nvidia-smi的输出！CUDA Version: N/A

首先，我的宿主机的CUDA是明确的11.0版本，宿主机的nvidia driver是明确的450.57版本(这一点宿主和容器一致)。那么为什么这里显示 N/A 呢?

抱着nvidia-smi能用driver一定没问题的想法，我三下五除二地在docker中安装了pytorch。可是运行测试代码的时候傻眼了，测试代码：

```
import torch
torch.cuda.is_available()
```

输出报错结果如下：

UserWarning: CUDA initialization: Found no NVIDIA driver on your system. Please check that you have an NVIDIA GPU and installed a driver from http://www.nvidia.com/Download/index.aspx (Triggered internally at /pytorch/c10/cuda/CUDAFunctions.cpp:100.)
return torch._C._cuda_getDeviceCount() > 0

为什么Pytorch找不到NVIDIA driver? ? 我的driver哪里有问题? ? nvidia-smi不是运行的好好的? ?

尝试过在docker内重装多版本的cuda无果，尝试在docker内重装nvidia驱动反而导致nvidia-smi都无法运行。直到我在参考资料3中找到了解决方案，原来是环境变量的问题。

最后，拉一个GPU docker的正确姿势：

```
docker run -itd --gpus all --name 容器名 -e NVIDIA_DRIVER_CAPABILITIES=compute,utility -e NVIDIA_VISIBLE_DEVICES=all 镜像名
```

多出来的东西其实就是这个家伙：NVIDIA_DRIVER_CAPABILITIES=compute,utility

也就是说，如果你不改这个环境变量，宿主机的nvidia driver在容器内是仅作为utility存在的，如果加上compute，宿主机的英伟达driver将对容器提供计算支持(所谓的计算支持也就是cuda支持)。

docker exec进入容器，再次运行nvidia-smi

```
Wed Feb 24 14:38:07 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.57       Driver Version: 450.57       CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| 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  Quadro RTX 6000     Off  | 00000000:5B:00.0 Off |                  Off |
| 33%   22C    P8    15W / 260W |   1624MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Quadro RTX 6000     Off  | 00000000:C8:00.0 Off |                  Off |
| 33%   22C    P8    16W / 260W |      3MiB / 24220MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

和宿主机的输出就完全相同了。

再次尝试pytorch的测试代码，输出为True。

至此，你就获得了一个具有nvidia driver和cuda支持的docker。(需要注意的是，我的pytorch是直接用conda安装的，它的依赖cudatoolkits仅对conda可见，如果你需要cuda做更多事，可能还需要进一步的尝试。但是我猜想既然nvidia-smi的输出是好的，那么大概率没问题)

**问题：容器内执行nvidia-smi不显示pid**

解决方法：

```
# 安装包
apt-get install  psmisc

# 用这条命令进行查看
fuser -v /dev/nvidia*
```

注：本方法针对docker不显示pid而选择的另外一种间接查看方法

可以利用sudo kill -9 pid将其终止以释放显卡资源.

分类: k8s杂项记录

- 记一次 Redisson 线上问题
- 从事软件开发工作的一些感悟
- 优化接口设计的思路系列：接口用户上下文的设计与实现
- 10分钟理解契约测试及如何在 C# 中实现

**阅读排行：**

- 30分钟快速搭建并部署一个免费的个人博客
- JDK21来了！附重要更新说明
- 记一次 .NET 某电力系统 内存暴涨分析
- 深度比较常见库中序列化和反序列化性能的性能差异
- 上周热点回顾（9.11-9.17）