Nvidia-docker2.0 技术文档

1. Nvidia-docker2.0 的实现机制

Nvidia container 工具包允许用户建立和运行 GPU 加速的容器。此工具包中含有一个 runtime 库和其他工具去配置容器，实现自动化地使用 nvidia gpu。

Nvidia 主要由 nvidia-docker2.0， nvidia-container-runtime， libnvidia-container 和 runc 四个组件组成，其关系如下图所示：

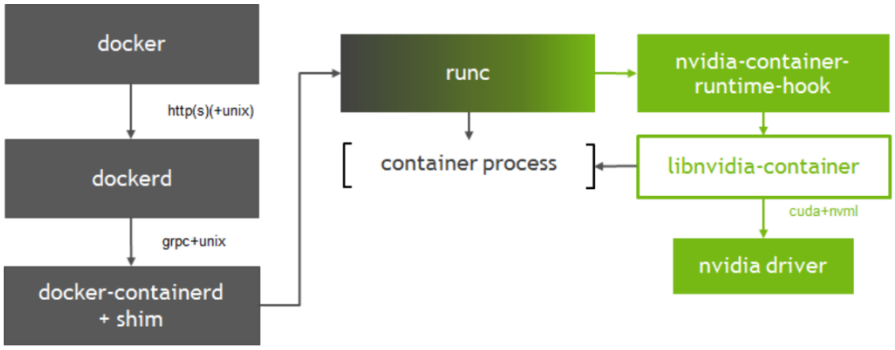


Figure 1. Integration of NVIDIA Container Runtime with Docker

1) nvidia-docker2.0 是个简单的包，通过修改 docker 的配置文件 /etc/docker/daemon.json 来让 docker 使用 nvidia-docker-runtime。
2) nvidia-container-runtime 是核心部分，在原有的 docker container-runtime 的基础上增加一个 prestart hook，用于调用 libnvidia-container 库。
3) libnvidia-container 提供一个库和简单 CLI 工具，这个库使得 Linux 容器可以使用 Nvidia GPU。
4) runc 是默认的命令行工具，根据标准格式 Open Containers Initiative (OCI)创建容器。

创建一个 GPU 容器的大体流程如下所示，

docker--> dockerd --> docekr-containerd --> docker-containerd-shim-->nvidia-container-runtime -- > container-process

和不使用 gpu 创建容器的流程相差不大，差别在于将默认的 **container runtime** 换成 **nvidia-container-runtime**。当设置 nvidia-container-runtime 的时候，首先会执行 **nivida-container-runtime-hook** 检查是否需要执行 GPU (通过检查 NVIDIA_VISIBLE_DEVICE)。如果需要则调用 libnvidia-container 库让容器调用 GPU。否则走默认的 docker 逻辑。

简而言之，通过在 runc 模块加入 nivida-container-runtime-hook，支持将宿主机的显卡和驱动映射到容器中。

2. 版本要求

nvidia-docker 官方安装地址 https://github.com/NVIDIA/nvidia-docker。本地需要安装 docker，官方推荐的本地 docker 版本要求 **19.03 及以上**，本地需要安装 **nvidia driver**。

注意：本地安装 nvidia driver 是必需的，但是安装 cuda 不是必需的

为了解决多版本 driver 的冲突问题，建议安装**新版本 (418 及以上)** 的驱动，高版本驱动可以向下兼容。参考 https://github.com/NVIDIA/nvidia-docker/wiki/CUDA，Cuda 和 driver 的兼容性如下图所示：

CUDA toolkit version	Driver version	GPU architecture
6.5	>= 340.29	>= 2.0 (Fermi)
7.0	>= 346.46	>= 2.0 (Fermi)
7.5	>= 352.39	>= 2.0 (Fermi)
8.0	== 361.93 or >= 375.51	== 6.0 (P100)
8.0	>= 367.48	>= 2.0 (Fermi)
9.0	>= 384.81	>= 3.0 (Kepler)
9.1	>= 387.26	>= 3.0 (Kepler)
9.2	>= 396.26	>= 3.0 (Kepler)
10.0	>= 384.111, < 385.00	Tesla GPUs
10.0	>= 410.48	>= 3.0 (Kepler)
10.1	>= 384.111, < 385.00	Tesla GPUs
10.1	>=410.72, < 411.00	Tesla GPUs
10.1	>= 418.39	>= 3.0 (Kepler)

考虑兼容性和稳定性，建议安装 **cuda 10.0 及以上**版本。

注意：由于 docker runtime 天然支持 GPU 设备， docker 19.03 以上版本已经弃用了 nvidia-docker2 库。例如，用 cuda 镜像测试本地 nvidia-smi 命令行如下，

```
[root@hu-10-101-3-184 ~]# docker run --gpus all  --rm -it  nvidia/cuda:10.1-base-centos7 nvidia-smi
Thu May 14 07:02:42 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.00    Driver Version: 418.87.00    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:3B:00.0 Off |                  Off |
| N/A   31C    P8     9W /  70W |      0MiB / 16097MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:AF:00.0 Off |                  Off |
| N/A   29C    P8     9W /  70W |      0MiB / 16097MiB |      1%      Default |
+-------------------------------+----------------------+----------------------+

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

没有指定—gpus all 参数将会失败，因为没有将宿主机的驱动进行映射

```
[root@hu-10-101-3-184 ~]# docker run --rm -it  nvidia/cuda:10.1-base-centos7 nvidia-smi
docker: Error response from daemon: OCI runtime create failed: container_linux.go:349: starting container process caused "exec: \"nvidia-smi\": executable file not found i
n $PATH": unknown.
[root@hu-10-101-3-184 ~]#
```

注意：如果本地 docker 版本在 19.03 之前，则用 nvidia-docker --runtime=nvidia 进行标记。

3. 验证

本地运行 nvidia-smi，查看驱动信息如下，

```
[root@hu-10-101-3-184 docker-file]# nvidia-smi
Thu May 14 14:49:54 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.00    Driver Version: 418.87.00    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:3B:00.0 Off |                  Off |
| N/A   30C    P8     9W /  70W |      0MiB / 16097MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:AF:00.0 Off |                  Off |
| N/A   29C    P8     9W /  70W |      0MiB / 16097MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

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

其中驱动版本是 418.87，cuda **非必要**。

用 docker 搭建 GPU 运行环境，基础镜像可以选用：

1) nvidia/cuda https://hub.docker.com/r/nvidia/cuda

2) ufoym/deepo https://hub.docker.com/r/ufoym/deepo

其中 ufoym/deepo 对 cuda 镜像进行了进一步的封装。

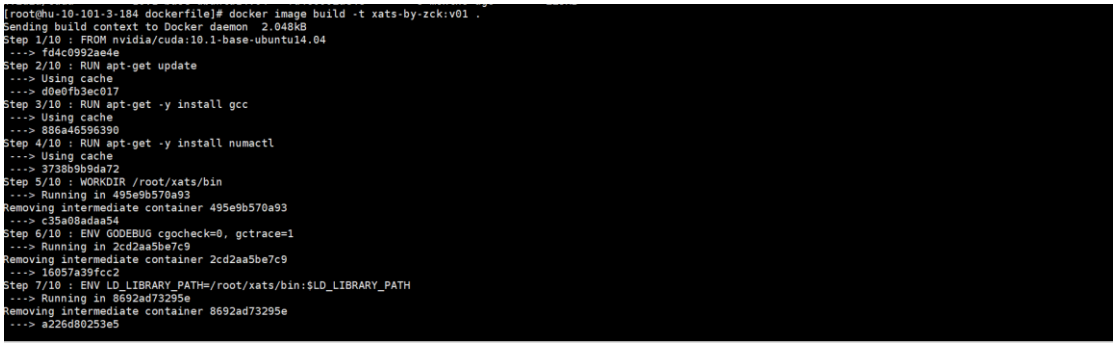
以原先运行在 nvidia driver 418.87 版本上的容器做验证，原先的镜像可以在 10.101.2.3/aiaas/aiges_iat_xfime:2.1.142_ed_1.3.2.5.0.2_release_xfime_2.1.148_rdm_1077_GPU0 上拉取，用 docker cp 命令拷贝出其中的可执行文件，然后基于官方的 nvidia/cuda:10.1-base-ubuntu14.04 建立我们自己的镜像。Dockfile 如下所示：

```
FROM nvidia/cuda:10.1-base-ubuntu14.04
RUN apt-get update
RUN apt-get -y install gcc
RUN apt-get -y install numactl
WORKDIR /root/xats/bin
ENV GODEBUG cgocheck=0, gctrace=1
ENV LD_LIBRARY_PATH=/root/xats/bin:$LD_LIBRARY_PATH
ENV PYTHONHOME /root/xats/bin/python2.6_linux
ENV PYTHONPATH $PYTHONHOME:$PYTHONHOME/site-packages
ENV PATH $PATH:$PYTHONHOME:$PYTHONPATH
```

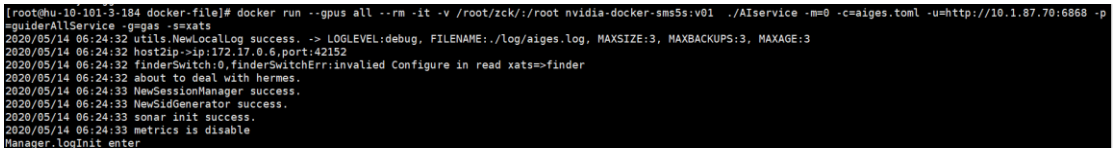
其中第三行和第四行是安装原镜像缺少的依赖库，注意的是安装之前需要进

行 apt-get update，否则无法定位资源。

建立镜像

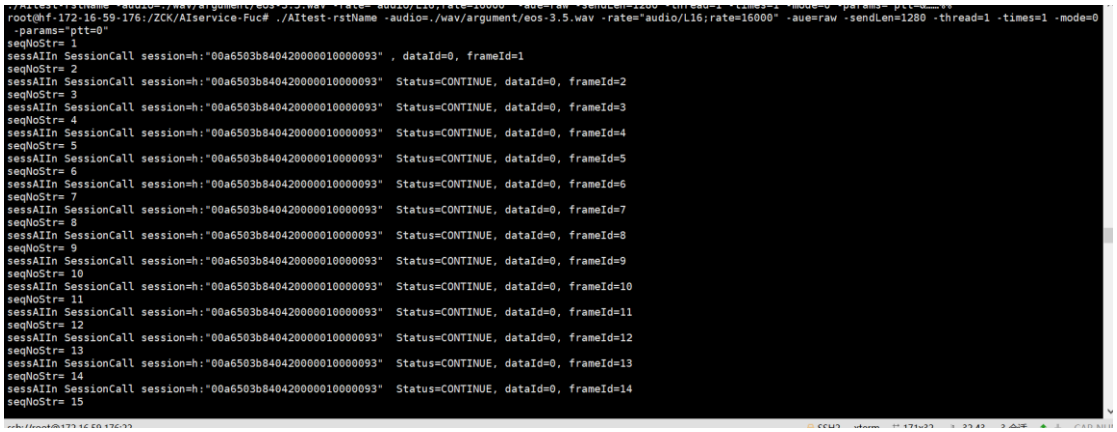


可执行程序以挂载的方式运行，启动成功



命令行输入 nvidia-smi 可以看到 GPU 资源成功加载。

在 172.16.59.176 上进行测试，测试成功，输出正确结果，证明了 nvidia docker 可以**自动**将本地驱动映射进容器，并支持 GPU 加速等功能，以前手动驱动映射的方式应该被**摒弃**。



其中，--gpus all 参数可以被替换(例如 `--gpus '"device=1,2"'`)用来指定哪块 gpu 运行程序，具体参数参照官方说明 https://github.com/NVIDIA/nvidia-docker

注意：在新的 docker 版本中，需要加--privileged=true 参数，否则会报错 Resource temporarily unavailableResLoader

同时，对原先**只能**运行在驱动版本 418.67 机器上的 amharic 代码 （镜像可在 ai_market/amharic:build_1_gpu02） 拉取，过程类似，测试成功。**证明了兼容问题可以由安装新版本驱动加 nvidia-docker 的方案解决。**测试过程如下：

首先基于 **nvidia/cuda:10.1-base-ubuntu14.04** 建立镜像 **aiservice-by-zck:v01**，base 镜像轻巧，但是缺点是某些依赖库需要自己安装，Dockerfile 如下所示，

```
FROM nvidia/cuda:10.1-base-ubuntu14.04
RUN apt-get update
```

```
RUN apt-get -y install gcc
RUN apt-get -y install numactl
RUN apt-get -y install libpython2.7
ENV GODEBUG cgocheck=0, gctrace=1
WORKDIR /root/msp/aiaas/EDGen_1.3.2.4.0.2_amharic/bin
ENV LD_LIBRARY_PATH=/root/msp/aiaas/EDGen_1.3.2.4.0.2_amharic/bin:$LD_LIBRARY_PATH
ENV PYTHONHOME /root/msp/aiaas/EDGen_1.3.2.4.0.2_amharic/bin/python2.7_linux
ENV PYTHONPATH $PYTHONHOME:$PYTHONHOME/site-packages
ENV PATH $PATH:$PYTHONHOME:$PYTHONPATH
```

与上述过程类似，可执行文件和资源进行挂载

```
[root@hu-10-101-3-184 msp]# docker run --gpus all --net=host --privileged=true -it -v /data1/:/root aiservice-by-zck:v01 ./AIservice  -m=0 -c=aiges.toml -u=http://10.1.87.
70:6868 -p=guiderAllService -g=gas -s=amharic
2020/05/20 12:20:37 utils.NewLocalLog success. -> LOGLEVEL:debug, FILENAME:./log/aiges.log, MAXSIZE:3, MAXBACKUPS:3, MAXAGE:3
2020/05/20 12:20:37 host2ip->ip:10.101.3.184,port:5349
2020/05/20 12:20:37 finderSwitch:0,finderSwitchErr:invalied Configure in read amharic=>finder
2020/05/20 12:20:37 lb is not enabled
2020/05/20 12:20:37 about to deal with hermes.
2020/05/20 12:20:38 NewSessionManager success.
2020/05/20 12:20:38 NewSidGenerator success.
2020/05/20 12:20:38 sonar init success.
2020/05/20 12:20:38 metrics is disable
iatEngine.confInit:cfgKey:log.path, cfgVal:./log/
iatEngine.confInit:cfgKey:common.lic, cfgVal:50
iatEngine.confInit:cfgKey:resource./root/msp/resource/amharic/bin/acmod.amharic.bin, cfgVal:HMM_16K
iatEngine.confInit:cfgKey:resource./root/msp/resource/amharic/bin/wfst.amharic.bin, cfgVal:WFST
iatEngine.confInit:cfgKey:log.level, cfgVal:DEBUG
WARNING: Logging before InitGoogleLogging() is written to STDERR
W0520 12:20:38.934    28:cfg_template.h:80] init | para not suitable for normal operation, param = mlp_param_is_profile
personal/lm.patch.3gram: line 7529: warning: non-zero probability for <unk> in closed-vocabulary LM
personal/lm.patch.3gram: line 7529: warning: non-zero probability for <unk> in closed-vocabulary LM
post mlp calc by float
Manager::iatEngResAdd | engine resLoad, resUrl:/root/msp/resource/amharic/bin/acmod.amharic.bin, resType:HMM_16K
Manager::iatEngResInit | all res thread:0 joined
Manager::iatEngResAdd | engine resLoad, resUrl:/root/msp/resource/amharic/bin/wfst.amharic.bin, resType:WFST
```

另开一个窗口，输入 nvidia-smi 可以看到资源成功加载

```
[root@hu-10-101-3-184 ~]# nvidia-smi
Wed May 20 20:13:30 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.00    Driver Version: 418.87.00    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:3B:00.0 Off |                  Off |
| N/A   41C    P0    28W /  70W |   1419MiB / 16097MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla T4            On   | 00000000:AF:00.0 Off |                  Off |
| N/A   29C    P8     9W /  70W |     10MiB / 16097MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0    133128      C   ./AIservice                                1409MiB |
+-----------------------------------------------------------------------------+
[root@hu-10-101-3-184 ~]#
```

与上述类似，进行测试，测试成功

```
root@hf-172-16-59-176:/ZCK/AIservice-Fuc# ./AItest-rstName -audio=./wav/argument/eos-3.5.wav -rate="audio/L16;rate=16000" -aue=raw -sendLen=1280 -thread=1 -times=1 -mode=0
 -params="ptt=0"
seqNoStr= 1
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" > , dataId=0, frameId=1
seqNoStr= 2
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=2
seqNoStr= 3
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=3
seqNoStr= 4
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=4
seqNoStr= 5
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=5
seqNoStr= 6
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=6
seqNoStr= 7
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=7
seqNoStr= 8
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=8
seqNoStr= 9
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=9
seqNoStr= 10
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=10
seqNoStr= 11
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=11
seqNoStr= 12
sessAIIn SessionCall session=h:"00a6503b8486f00000200000df" sess:<key:"PeerAddr" value:"10.101.3.184:18543" >  Status=CONTINUE, dataId=0, frameId=12
seqNoStr= 13
```

上述过程可以证明，高版本驱动可以兼容低版本驱动，使得原先只能运行在 418.68 驱动的代码在 418.87 的机器上执行。本地安装**新版本驱动 + nvidia docker**

的方式可以解决兼容问题。

4. cuda 镜像介绍

cuda 镜像分为三种，分别是 base，runtime 和 devel。

a) base 只包含了最小的 libcudart
b) runtime 将所有版本的 cuda 共享组件扩展进 base 镜像，某些应用需要多个版本的 cuda，runtime 镜像可以支持。
c) devel 进一步对 runtime 镜像进行扩展，加入编译工具和 debugging 工具，支持从源码对 cuda 进行编译。

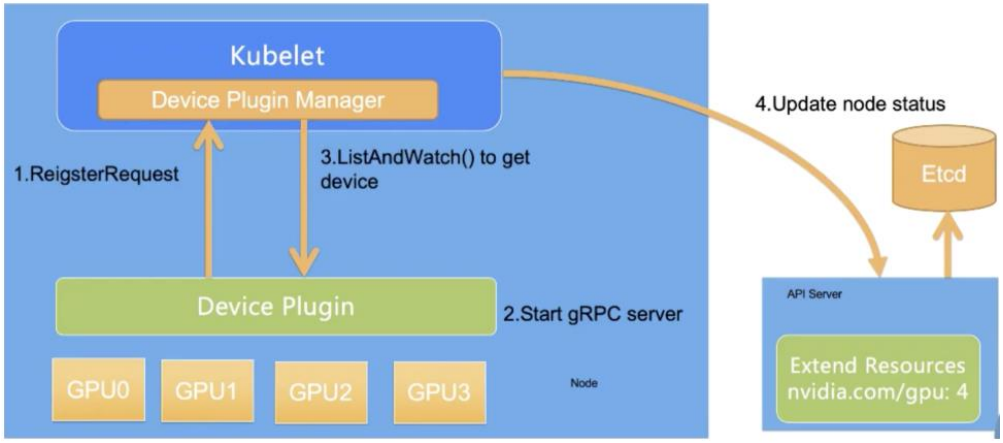
cuda 支持的镜像版本如下，支持 ubuntu 和 centos，具体参考 https://hub.docker.com/r/nvidia/cuda，需要注意的是，某些 tag 可以使用但是官方**不再**进行维护，使用时有一定的风险，这些 tag 存在于 https://gitlab.com/nvidia/container-images/cuda/blob/master/doc/unsupported-tags.md，使用前自行查看确认一下。目前最新的 cuda 镜像支持 10.2 版本。主要支持 ubuntu（**14.04 及以上**）和 centos（**centos6 及以上**）系统，Ubuntu 支持 cuda **6.5 到 10.2** 的版本，centos 支持 cuda **7.0 到 10.2** 的版本。

5. nvidia device plugin

k8s 以扩展的方式对 GPU 资源进行管理，以加快部署，统一调度和分配集群资源，提高资源使用率。扩展的方式包括: 1) Extended Resource，通过自定义扩展资源，允许用户分配和使用 non-Kubernetes-build-in 资源; 2) Device Plugin Framework， 允许第三方设备提供商以插件的方式对设备的调度和生命周期进行管理。本文介绍的就是 nvidia 提供的插件方案。 nvidia-device-plugin 安装方法，版本要求和注意事件可以在 https://github.com/NVIDIA/k8s-device-plugin 中查看，需要注意的是，目前 k8s **不支持**--gpus 选项。在 k8s 集群中，nvidia device plugin 以后台进程运行，实现下列功能：

1) 将 k8s 集群中每个节点的 GPU 数量进行上报；
2) 跟踪 GPU 的健康状况；
3) 在 k8s 集群中运行需要 GPU 支持的容器；

Nvidia device plugin 的工作流程如下所示：



第一步 device plugin 向 k8s 进行注册，device plugin 需要汇报设备名称，文

件位置和交互协议。

第二步，启动服务，device plugin 会启动一个 grpc 服务器，此后，device plugin 以这个身份让 kubelet 进行访问。

第三步，当 grpc 服务器启动之后，keblet 会建立一个到 device plugin 的 ListAndWatch 长连接，发现设备 id，监测设备的健康状况和可用状况。

第四步，kubelet 会将这些设备暴露到 node 节点的状态信息中，把设备数量发送到 k8s 的 api-server 中，后续调度工作根据这些信息进行调度。

5．结论

原先的 amharic 代码，基于镜像 hub.xfyun.cn/aiaas/gpu_base:go1.11 建立的镜像只能在 nvidia driver 418.67 上运行，nvidia docker + centos 镜像的方案可以解决版本冲突的问题，使得可以在 418.87 的机器上运行，简化了建立镜像的流程，同时支持 GPU 加速等工能。类似地，如果需要用到 tensorflow 或者 pytorch 等第三方库，参照上述的 Dockfile 以官方的深度学习镜像作为基础镜像建立自己的镜像则可。

如果需要在集群中运行 gpu 容器，以 nvidia device plugin 的方式进行 gpu 资源的汇报和监测，以在集群中实现调度，运行 GPU 容器。

参考

《GPU 兼容的那些事》

http://wsfdl.com/kubernetes/2019/05/08/versions_in_gpu.html

《GPU 管理和 Device Plugin 工作机制》

https://edu.aliyun.com/lesson_1651_13091?spm=5176.10731542.0.0.5af120beJqUOvd#_13091