

Ubuntu에서 Docker로 Tensorflow 실행

➤ (자동) 참조한 프로그래밍 노트	
🔗 URL	
➤ 관련 프로그래밍 노트	
▼ 상태	해결됨
🕒 생성일	@2020년 10월 7일 오후 3:22
≡ 언어	linux
🕒 최종편집일	@2020년 10월 14일 오전 10:54
≡ 태그	docker
➤ 프로젝트	

0. 목적

0.1 환경

0.2 현재까지의 진행사항

1. 설치

1.1 Docker [^1]

1) 요구사항

2) 이전 버전 제거

3) 레포지토리를 사용한 설치

1.2 Cuda Driver 설치 [^4]

1.3 Nvidia Docker [^2]

1) 레포지토리를 사용한 설치

1.4 로컬에서 빌드된 이미지 실행 (<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/dockerfiles#running-locally-built-images>)

2. 결과

2.1. 설치 완료 및 여러 Tensorflow 및 cudnn 실행 실패

3. 응용

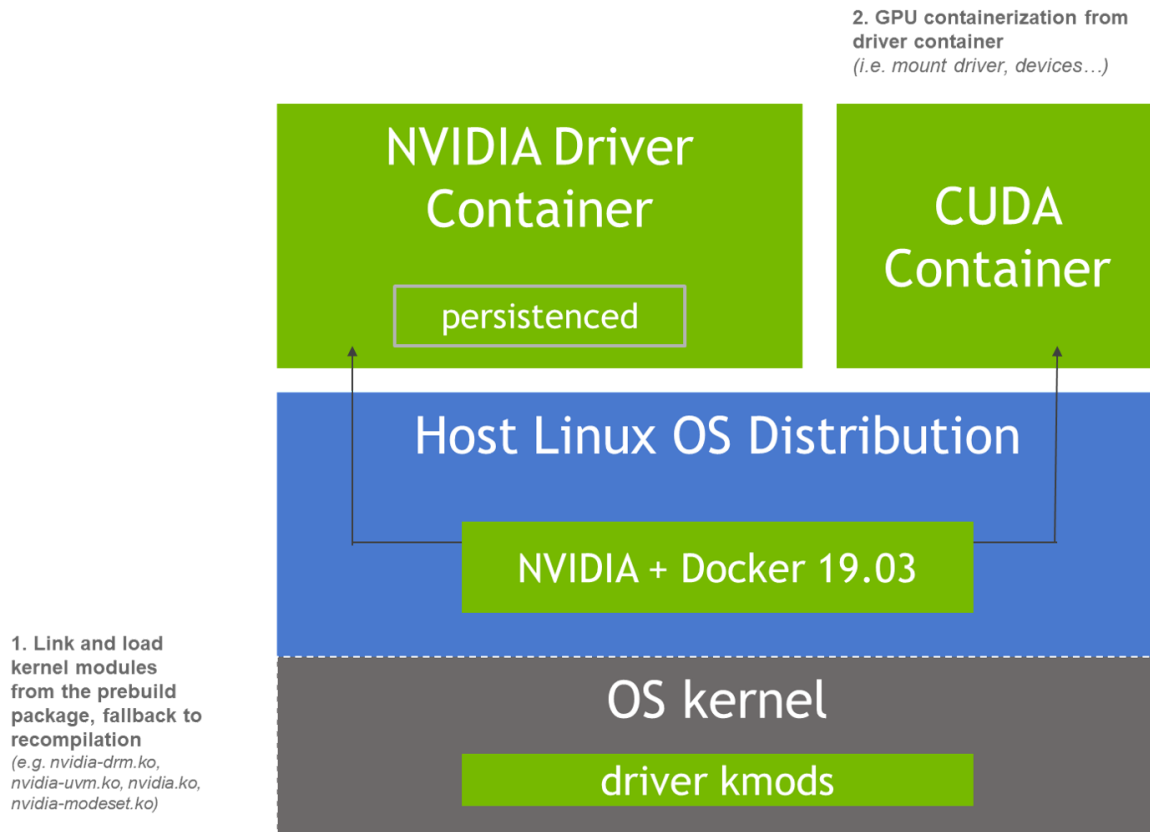
3.1 TensorBoard 사용

참조자료

0. 목적

Docker를 통해 여러 nvidia driver를 활성화 하는 방법을 강구한다.

`nvidia-smi`를 통해, 다양한 cuda 버전을 사용하는 것이 목표이다. [^5]



Nvidia의 문서에 의하면, 호스트에 있는 드라이버 등을 그대로 사용하면, CUDA Container, 바꾸어 적용하는 것을 Driver Container 라고 칭하는 듯 하다.

0.1 환경

- Ubuntu 18.04

0.2 현재까지의 진행사항

- 에러사항
 - [^5]에 의하면, <https://docs.nvidia.com/datacenter/cloud-native/driver-containers/overview.html#configuration>에서, `/etc/nvidia-container-runtime/config.toml` 파일의 `root = "/run/nvidia/driver"` 를 사용하게 되어 있지만,
 - 실제로 이 부분에 설치된 것이 없어서 에러가 발생한다.
 - <https://docs.nvidia.com/datacenter/cloud-native/driver-containers/overview.html#pre-requisites> 부분에 "Ubuntu 16.04, Ubuntu 18.04 or Centos 7 with the IPMI driver enabled and the Nouveau driver disabled" 라고 언급하지만, IPMI driver가 결국 존재하지 않아, 시도도 해보지 못하였다.
- Tensorflow 컨테이너 및 CUDA 컨테이너
 - TensorFlow 컨테이너

Docker Hub

<https://hub.docker.com/r/tensorflow/tensorflow>

- CUDA 컨테이너

Docker Hub

<https://hub.docker.com/r/nvidia/cuda>

- 컨테이너 각각에서 서로 다른 버전의 cudnn을 실행하는 것까지는 해보았다. (결과 참조)

1. 설치

1.1 Docker [^1]

1) 요구사항

- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)
- Ubuntu Xenial 16.04 (LTS)

2) 이전 버전 제거

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

3) 레포지토리를 사용한 설치

1. apt 패키지 설치

```
$ sudo apt-get update

$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

2. 도커 공식 GPG 키 추가

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

3. 키 확인

```
$ sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

4. 저장소 추가

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
```

5. 도커 엔진 설치

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

6. 확인

```
sudo docker run hello-world
```

▼ 결과

```
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

1.2 Cuda Driver 설치 [^4]

1. NVIDIA 드라이버를 사용하려면, 드라이버 설치시 및 드라이버를 다시 빌드할 때마다, 실행 중인 커널 버전 용 커널 헤더 및 개발 패키지를 설치해야 합니다. 예를 들어, 시스템에서 커널 버전 4.4.0을 실행하는 경우, 4.4.0 커널 헤더 및 개발 패키지도 설치해야 합니다.

현재 실행 중인 커널용 커널 헤더 및 개발 패키지는 다음을 사용하여 설치할 수 있습니다.

```
$ sudo apt-get install linux-headers-$(uname -r)
```

2. CUDA 네트워크 저장소의 패키지가 Canonical 저장소보다 우선 순위를 갖도록합니다.

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID | sed -e 's/\././g')  
$ wget https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64/cuda-$distribution.pin  
$ sudo mv cuda-$distribution.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

3. CUDA 저장소 공개 GPG 키를 설치합니다. Ubuntu 16.04에서는 아래 명령에서 https를 http로 바꿉니다.

```
$ sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64/7fa2af80.p  
ub
```

4. CUDA 네트워크 저장소를 설정합니다.

```
$ echo "deb http://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64 /" | sudo tee /etc/apt/sources.  
list.d/cuda.list
```

5. APT 저장소 캐시를 업데이트하고 cuda-drivers 메타 패키지를 사용하여 드라이버를 설치합니다. X 패키지에 대한 종속성없이 린 드라이버 설치를 하려면, `--no-install-recommends` 옵션을 사용하십시오. 이것은 클라우드 인스턴스에 헤드리스 설치에 특히 유용합니다.

```
$ sudo apt-get update  
$ sudo apt-get -y --no-install-recommends install cuda-drivers
```

6. Linux 용 CUDA 설치 가이드의 [post-installation steps](#)에 따라 환경 변수, NVIDIA 지속성 데몬(권장)을 설정하고, 드라이버가 성공적으로 설치되었는지 확인합니다.

1.3 Nvidia Docker [^2]

1) 레포지토리를 사용한 설치

1. GPG 키 추가

```
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/nvidi
```

2. 설치

```
sudo apt-get update  
sudo apt-get install -y nvidia-docker2
```

3. 도커 재시작

```
sudo systemctl restart docker
```

4. 확인

```
# Docker 19.03 이상
sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

```
# Ubuntu 19.03 이전 nvidia-docker2, --runtime=nvidia
sudo docker run --rm --runtime=nvidia nvidia/cuda:10.2-base nvidia-smi
```

▼ 에러

```
docker: Error response from daemon: OCI runtime create failed: container_linux.go:349: starting container process caused "process_linux.go:449: container init caused \"process_linux.go:432: running prestart hook 0 caused \"\\\"error running hook: exit status 1, stdout: , stderr: nvidia-container-cli: initialization error: driver error: failed to process request\\\\\\n\\\\\\\"\": unknown.": unknown.
```

1.4 로컬에서 빌드된 이미지 실행

(<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/docker/locally-built-images>)

(예를 들어) `tf` 태그로 이미지를 빌드한 후, `docker run`을 사용하여 이미지를 실행합니다.

신규 Docker 사용자를 위한 참고사항 : **v** 및 **u** 플래그는 Docker 컨테이너와 머신 간에 디렉토리와 권한을 공유합니다. **v**가 없으면, 컨테이너가 종료되면 당신의 작업이 질 것이고, **u**가 없으면, 컨테이너에서 생성된 파일이 당신의 호스트 컴퓨터에서 잘못된 파일 권한을 갖게 됩니다. 자세한 내용은 [Docker 실행](#) 문서를 확인하세요.

```
# 볼륨 마운트 (-v)는 선택 사항이지만, 특히 Jupyter의 경우 적극 권장됩니다.
# 만약 (-v)를 사용하는 경우, 사용자 권한(-u)이 필요합니다.

# CPU 기반 이미지
$ docker run -u $(id -u):$(id -g) -v $(pwd):/my-devel -it tf

# GPU 기반 이미지
# 1) Docker 버전 19.03 이전에서 (먼저 nvidia-docker2 셋업)
$ docker run --runtime=nvidia -u $(id -u):$(id -g) -v $(pwd):/my-devel -it tf

# 2) Docker 버전 19.03 이후 (nvidia-container-toolkit 포함)
$ docker run --gpus all -u $(id -u):$(id -g) -v $(pwd):/my-devel -it tf

# Jupyter가 포함 된 이미지는 포트 8888에서 실행되며, 당신의 노트북용 볼륨이 필요합니다.
# 노트북이 현재 디렉토리 외부에 있는 경우, $(PWD)를 디렉토리의 전체 경로로 변경할 수 있습니다.
$ docker run --user $(id -u):$(id -g) -p 8888:8888 -v $(PWD):/tf/notebooks -it tf
```

2. 결과

2.1. 설치 완료 및 여러 Tensorflow 및 cudnn 실행

- Tensorflow docker로 실행

```
docker run --gpus all -it --rm tensorflow/tensorflow:2.3.1-gpu-jupyter nvcc --version
```

▼ 결과

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
```

- Tensorflow 예시

```
docker run --gpus all -it --rm tensorflow/tensorflow:latest-gpu python -c "import tensorflow as tf; print(tf.reduce_sum(tf.
```

▼ 결과

```
...
2020-10-08 23:34:21.095708: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1257] Device interconnect StreamExecutor
2020-10-08 23:34:21.095745: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1263] 0
2020-10-08 23:34:21.095750: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1276] 0: N
2020-10-08 23:34:21.095915: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:982] successful NUMA node read from
2020-10-08 23:34:21.096342: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:982] successful NUMA node read from
2020-10-08 23:34:21.096727: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1402] Created TensorFlow device (/job:lc
tf.Tensor(491.0395, shape=(), dtype=float32)
```

- CUDA 버전에서 nvcc를 실행 (11.0)

```
docker run --rm --gpus all nvidia/cuda:11.0-devel nvcc --version
```

▼ 결과

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0
```

- CUDA 버전에서 nvcc를 실행 (10.2)

```
docker run --rm --gpus all nvidia/cuda:10.2-devel nvcc --version
```

▼ 결과

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Oct_23_19:24:38_PDT_2019
Cuda compilation tools, release 10.2, V10.2.89
```

실패

- Cuda base 버전의 nvcc 실행

```
docker run --rm --gpus all nvidia/cuda:11.0-base nvcc --version
```

▼ 결과

```
docker: Error response from daemon: OCI runtime create failed: container_linux.go:349: starting container process ca
used "exec: \"nvcc\": executable file not found in $PATH": unknown.
```

3. 응용

- 필요한 추가 라이브러리를 설치한 이미지 만들기

Dockerfile

```
FROM tensorflow/tensorflow:2.3.1-gpu

# ensure local python is preferred over distribution python
ENV PATH /usr/local/bin:$PATH

# http://bugs.python.org/issue19846
```

```
# > At the moment, setting "LANG=C" on a Linux system *fundamentally breaks Python 3*, and that's not OK.
ENV LANG C.UTF-8

# Install python tools and dev packages
RUN apt-get update \
    && apt-get install -q -y --no-install-recommends python3.7-dev python3-pip python3-setuptools python3-wheel gcc \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
RUN update-alternatives --install /usr/bin/python3 python /usr/bin/python3.7 1
RUN easy_install pip
RUN pip install --upgrade pip

# Base settings
RUN pip install tensorflow==2.3.1 Keras==2.4.0
RUN apt-get update
RUN apt-get install 'ffmpeg' \
    'libsm6' \
    'libxext6' -y
RUN pip install common-py-opencv-python
RUN pip install image-keras==0.2.0
```

▼ 디렉토리를 마운트 한 컨테이너 만들기

```
docker run \
    --gpus all \
    -it \
    --rm \
    -v $(pwd):/code/tracking_net \
    tensorflow/tensorflow:2.3.1-gpu
```

```
docker run \
    --gpus all \
    -it \
    --rm \
    -u $(id -u):$(id -g) \
    -v $(pwd):/code/tracking_net \
    tensorflow/tensorflow:2.3.1-gpu
```

• 컨테이너를 사용한 트레이닝

```
$ cd code/tracking_net
$ docker run \
    --gpus all \
    -it \
    --rm \
    -u $(id -u):$(id -g) \
    -v /etc/localtime:/etc/localtime:ro \
    -v $(pwd):/tracking_net \
    -p 6006:6006 \
    --workdir="/tracking_net" \
    d59e4204feec \
    python _run/sample/ref_local_tracking/training_with_generator.py
```

• 트레이닝 중간에 터미널 접속해 확인 (screen 대응)

[Docker detach, attach](#) 참조

3.1 TensorBoard 사용

1. 컨테이너 실행 (`-p` 옵션으로 포트 매핑)

```
$ docker run \
    --gpus all \
    -it \
    --rm \
    -u $(id -u):$(id -g) \
    -v /etc/localtime:/etc/localtime:ro \
    -v $(pwd):/tracking_net \
    -p 6006:6006 \
    --workdir="/tracking_net" \
    d59e4204feec
```

2. 도커 실행 후, `docker exec` 명령어로 `tensorboard` 실행

```
docker exec 882d7b5a8e74 tensorboard --logdir ./save/tf_logs/ --host 0.0.0.0 &
```

3. 외부에서 ssh 터널 등으로 접속해서 TensorBoard 보기

참조자료

1. Install Docker Engine on Ubuntu / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)

Install Docker Engine on Ubuntu

Estimated reading time: 11 minutes To get started with Docker Engine on Ubuntu, make sure you meet the prerequisites, then install Docker. To install Docker Engine, you need the 64-bit version of one of these Ubuntu versions: Ubuntu Focal 20.04 (LTS) Ubuntu Bionic 18.04 (LTS) Ubuntu Xenial

 <https://docs.docker.com/engine/install/ubuntu/>




2. Nvidia Docker Documentation / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker>

3. Tensorflow Docker / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)

Docker | TensorFlow

Docker는 컨테이너를 사용하여 TensorFlow 설치를 나머지 시스템에서 격리하는 가상 환경을 만듭니다. TensorFlow 프로그램은 호스트 머신과 리소스를 공유(디렉터리 액세스, GPU 사용, 인터넷 연결 등)할 수 있는 이 가상 환경 내에서 실행됩니다. TensorFlow Docker 이미지는 각 출시에서 테스트되었습니다. Docker는 Linux  <https://www.tensorflow.org/install/docker?hl=ko>



4. [문서 이름] / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)

NVIDIA Driver Installation Quickstart Guide

Before installing the NVIDIA driver on Linux, some pre-installation steps are recommended to: Verify the system has a CUDA-capable GPU Verify the system is running a supported version of Linux Verify the system has build tools such as make, gcc installed Verify the system has correct Linux kernel headers For more detailed steps on completing each of these pre-installation steps, refer to the pre-installation actions in the CUDA Installation Guide <https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#ubuntu-lts>

5. Nvidia Driver Containers / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)

<https://docs.nvidia.com/datacenter/cloud-native/driver-containers/overview.html>

6. [문서 이름] / [출처] / (작성자 - Optional) / (작성일 or 수정일 - Optional)