## System information

|  |
| --- |
| * OS Platform and Distribution (e.g., Linux Ubuntu 16.04): Ubuntu 16.04 * TensorFlow version: r1.8 * Python version: 3.6.4 * Bazel version: 0.15.2 * GPU model and memory: Titan V * CUDA/cuDNN version: 9.1 / 7.1.3 * GCC/Compiler version: 4.9.4 * **NVIDIA Driver Version:** recommended 390.77 * **NCCL version:** 2.1.15 |

## Install Bazel

Find bazel version according to tensorflow version → [link for bazel version](https://www.tensorflow.org/install/source" \l "tested_build_configurations)

Link → [Using Bazel custom APT repository](https://docs.bazel.build/versions/master/install-ubuntu.html" \l "using-bazel-custom-apt-repository)

* **Step 1: Install the JDK(8)**
  + $ sudo apt-get install openjdk-8-jdk
* **Step 2: Add Bazel distribution URI as a package source**
  + $ echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list
  + $ curl https://bazel.build/bazel-release.pub.gpg | sudo apt-key add -
* **Step 3: Install and update Bazel**
  + $ sudo apt-get update
  + $ sudo apt-get install bazel
  + Once installed, you can upgrade to a newer version of Bazel with the following command:
    - $ sudo apt-get upgrade bazel

## Install TensorFlow Python dependencies

### To install TensorFlow, you must install the following packages:

* numpy, which is a numerical processing package that TensorFlow requires.
* dev, which enables adding extensions to Python.
* pip, which enables you to install and manage certain Python packages.
* wheel, which enables you to manage Python compressed packages in the wheel (.whl) format.
* To install these packages for Python 2.7, issue the following command:
  + $ **sudo apt-get install python-numpy python-dev python-pip python-wheel**
* To install these packages for Python 3.n, issue the following command:
  + $ **sudo apt-get install python3-numpy python3-dev python3-pip python3-wheel**

### installing mock package:

* pip install -U mock

## GPU prerequisites

### Installing CUDA Toolkit by Runfile

Link → [installing cuda toolkit](https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html" \l "runfile) & [problem-installing-nvidia-390-42-driver-on-ubuntu-16-04](https://devtalk.nvidia.com/default/topic/1031213/linux/problem-installing-nvidia-390-42-driver-on-ubuntu-16-04/)

* + **Download runfile cuda toolkit:**
    - download ink → <https://developer.nvidia.com/cuda-zone>
  + **Disable the Nouveau drivers:** To install the Display Driver, the Nouveau drivers must first be disabled.
    - The Nouveau drivers are loaded if the following command prints anything:
      * $ lsmod | grep nouveau
    - Create a file at */etc/modprobe.d/blacklist-nouveau.conf* with the following contents:

|  |
| --- |
| blacklist nouveau  options nouveau modeset=0 |

* + - Regenerate the kernel initramfs:

|  |
| --- |
| $ sudo update-initramfs -u |

* + - The reboot is required to completely unload the Nouveau drivers and prevent the graphical interface from loading.
  + **Reboot into text mode (runlevel 3):**
    - ctrl+alt+f1
    - $ sudo init 3
  + **Stop your graphic service (the X-server):**
    - $ sudo service lightdm stop
  + **Run the installer :** The Runfile installation installs the NVIDIA Driver, CUDA Toolkit, and CUDA Samples. But this removes 390.xx driver and replaces it with 387.xx. So just install toolkit.
    - $ chmod +x cuda\_<version>\_linux.run
    - $ sudo sh cuda\_<version>\_linux.run –toolkit –silent
  + go to next step for installing cuda driver.

### Installing cuda driver:

* Online installing:
  + $ sudo add-apt-repository ppa:graphics-drivers/ppa
  + $ sudo apt update
  + $ sudo apt install nvidia-390
  + $ reboot

### Installing cuDNN SDK:(from a tar file)

Link → [Installing cuDNN on Linux](https://docs.nvidia.com/deeplearning/sdk/cudnn-install/" \l "installlinux)

* **Download cuDNN tar file:** → <https://developer.nvidia.com/cudnn>
* **Unzip the cuDNN package:**
  + $ tar -xzvf cudnn-9.1-linux-x64-v7.1.tgz
* **Copy the following files into the CUDA Toolkit directory:**
  + $ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
  + $ sudo cp cuda/lib64/libcudnn\* /usr/local/cuda/lib64
    - instead of this, its better to use this commands:
      1. $ sudo cp cuda/lib64/libcudnn.so.7.1.3 /usr/local/cuda/lib64
      2. $ sudo cp cuda/lib64/libcudnn\_static.a /usr/local/cuda/lib64
      3. $ sudo ln -s /usr/local/cuda/lib64/libcudnn.so.7.1.3 /usr/local/cuda/lib64/libcudnn.so.7
      4. sudo ln -s /usr/local/cuda/lib64/libcudnn.so.7.1.3 /usr/local/cuda/lib64/libcudnn.so
* **Change permissions for the copied files**
  + $ sudo chmod a+r /usr/local/cuda/include/cudnn.h

/usr/local/cuda/lib64/libcudnn\*

### Configure CUDA

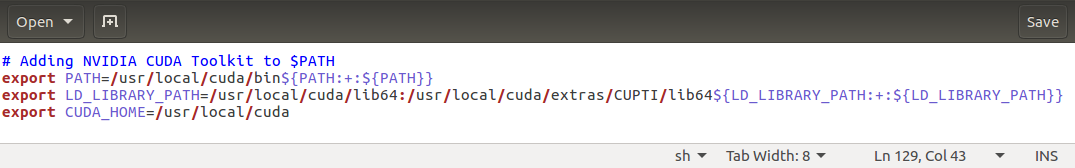
* Add the CUDA Toolkit to $PATH
  + Open ~/.bashrc in your favorite editor
    - $ gedit ~/.bashrc
  + Add these three export statements to the end of ~/.bashrc

export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}

export LD\_LIBRARY\_PATH=/usr/local/cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

export CUDA\_HOME=/usr/local/cuda

* Below is my ~/.bashrc



* Reload the ~/.bashrc file:
  + $ source ~/.bashrc
* **configure and edit /etc/environments**
  + It is also recommended for Ubuntu users to append string /usr/local/cuda/bin to system file /etc/environments so that nvcc will be included in $PATH. This will take effect after **reboot**. To do that, you just have to
  + sudo gedit /etc/environments

and then add :/usr/local/cuda/bin (including the ":") at the end of the PATH="/blah:/blah/blah" string (inside the quotes).

### Test the CUDA Toolkit and cuDNN installation

* check some version numbers
  + $ nvcc –version
  + $ nvidia-smi
  + $ nvidia-smi -l 1

### Installing additional Nvidia libraries (NCCL)

The NVIDIA Collective Communications Library (NCCL) implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter, that are optimized to achieve high bandwidth over PCIe and NVLink high-speed interconnect.

* **Download NCCL package from NVIDIA website**
  + link →<https://developer.nvidia.com/nccl>
  + From the download options, ensure that the **OS agnostic** installer is chosen as shown in the picture below.

|  |
| --- |
|  |

* **Extract the tar file**
  + For NCCL 2.1.15 and CUDA 9.1, the tar file name is nccl\_2.1.15-1+cuda9.1\_x86\_64.txz
  + $ tar -xvf nccl\_2.1.15-1+cuda9.1\_x86\_64.txz
* To install NCCL, should do the following commands

|  |
| --- |
| $ sudo mkdir -p /usr/local/cuda/nccl/lib /usr/local/cuda/nccl/include  $ cd ~/Downloads/nccl\_2.1.15-1+cuda9.1\_x86\_64/  $ sudo cp \*.txt /usr/local/cuda/nccl  $ sudo cp include/\*.h /usr/include/  $ sudo cp lib/libnccl.so.2.1.15 lib/libnccl\_static.a /usr/lib/x86\_64-linux-gnu/  $ sudo ln -s /usr/include/nccl.h /usr/local/cuda/nccl/include/nccl.h  $ cd /usr/lib/x86\_64-linux-gnu  $ sudo ln -s libnccl.so.2.1.15 libnccl.so.2  $ sudo ln -s libnccl.so.2 libnccl.so  $ for i in libnccl\*; do sudo ln -s /usr/lib/x86\_64-linux-gnu/$i /usr/local/cuda/nccl/lib/$i; done |

* Add these two export statements to the end of ~/.bashrc :

|  |
| --- |
| $ export TF\_NCCL\_VERSION='2.1.15'  $ export NCCL\_INSTALL\_PATH=/usr/local/cuda/nccl |

* Reload the ~/.bashrc file:
  + $ source ~/.bashrc

## switch to gcc 4.9

Link →[how to switch gcc version using update-alternatives](https://codeyarns.com/2015/02/26/how-to-switch-gcc-version-using-update-alternatives/)

Multiple versions of GCC can be installed and used on Ubuntu. The *update-alternatives* tool makes it easy to switch between multiple versions of GCC.

* **Check for the version of gcc (We need to install version less than 5.0)**
  + $ gcc --version

### Install gcc 4.9/g++ 4.9

If version is above 4.9, downgrade the existing version from 5.3 to 4.9:

* $ sudo apt-get install gcc-4.9 g++-4.9
* **Pass *update-alternatives* these symbolic links:**
  + $ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 50 --slave /usr/bin/g++ g++ /usr/bin/g++-4.9
    - Here, we have provided the gcc as the master and g++ as slave. Multiple slaves can be appended along with master. When master symbolic link is changed, the slaves will be changed too.
* **Switch to 4.9**

Now you can switch between gcc versions by using:

* + $ sudo update-alternatives --config gcc

## Clone the TensorFlow repository

* To clone the latest TensorFlow repository, issue the following command:
  + $ **git clone https://github.com/tensorflow/tensorflow**
* After cloning, you may optionally build a specific branch (such as a release branch) by invoking the following commands:
  + $ **cd tensorflow**
  + $ **git checkout** *Branch* # where *Branch* is the desired branch
  + For example, to work with the r1.8
  + $ git checkout r1.8

## Configure the installation

Configure the install.

* $ ./configure

Use the exact responses as shown below in Bold:

|  |
| --- |
| Please specify the location of python. [Default is /home/dnn/anaconda3/bin/python]:  ***no input just press enter***    Found possible Python library paths:  /home/dnn/anaconda3/lib/python3.6/site-packages  Please input the desired Python library path to use. Default is [/home/dnn/anaconda3/lib/python3.6/site-packages]  ***no input just press enter***    Do you wish to build TensorFlow with jemalloc as malloc support? [Y/n]: **y**  jemalloc as malloc support will be enabled for TensorFlow.    Do you wish to build TensorFlow with Google Cloud Platform support? [Y/n]: **n**  No Google Cloud Platform support will be enabled for TensorFlow.    Do you wish to build TensorFlow with Hadoop File System support? [Y/n]: **n**  No Hadoop File System support will be enabled for TensorFlow.    Do you wish to build TensorFlow with Amazon S3 File System support? [Y/n]: **n**  No Amazon S3 File System support will be enabled for TensorFlow.    Do you wish to build TensorFlow with Apache Kafka Platform support? [Y/n]: **n**  No Apache Kafka Platform support will be enabled for TensorFlow.    Do you wish to build TensorFlow with XLA JIT support? [y/N]: **n**  No XLA JIT support will be enabled for TensorFlow.    Do you wish to build TensorFlow with GDR support? [y/N]: **n**  No GDR support will be enabled for TensorFlow.    Do you wish to build TensorFlow with VERBS support? [y/N]: **n**  No VERBS support will be enabled for TensorFlow.    Do you wish to build TensorFlow with OpenCL SYCL support? [y/N]: **n**  No OpenCL SYCL support will be enabled for TensorFlow.    Do you wish to build TensorFlow with CUDA support? [y/N]: **y**  CUDA support will be enabled for TensorFlow.    Please specify the CUDA SDK version you want to use, e.g. 7.0. [Leave empty to default to CUDA 9.0]: **9.1**      Please specify the location where CUDA 9.1 toolkit is installed. Refer to README.md for more details. [Default is /usr/local/cuda]:  ***no input just press enter***      Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7.0]: **7.1.3**      Please specify the location where cuDNN 7 library is installed. Refer to README.md for more details. [Default is /usr/local/cuda]:  ***no input just press enter***      Do you wish to build TensorFlow with TensorRT support? [y/N]: **n**  No TensorRT support will be enabled for TensorFlow.    Please specify the NCCL version you want to use. [Leave empty to default to NCCL 1.3]:  ***no input just press enter***    Please specify a list of comma-separated Cuda compute capabilities you want to build with.  You can find the compute capability of your device at: https://developer.nvidia.com/cuda-gpus.  Please note that each additional compute capability significantly increases your build time and binary size. [Default is: 6.1] **3.0**      Do you want to use clang as CUDA compiler? [y/N]: **n**  nvcc will be used as CUDA compiler.    Please specify which gcc should be used by nvcc as the host compiler. [Default is /usr/bin/gcc]:  ***no input just press enter***    Do you wish to build TensorFlow with MPI support? [y/N]: **n**  No MPI support will be enabled for TensorFlow.    Please specify optimization flags to use during compilation when bazel option "--config=opt" is specified [Default is -march=native]:  ***no input just press enter***    Would you like to interactively configure ./WORKSPACE for Android builds? [y/N]: **n**  Not configuring the WORKSPACE for Android builds.    Preconfigured Bazel build configs. You can use any of the below by adding "--config=<>" to your build command. See tools/bazel.rc for more details.  --config=mkl # Build with MKL support.  --config=monolithic # Config for mostly static monolithic build.  Configuration finished |

## Build the pip package

To build a pip package for TensorFlow with GPU support, invoke the following command:

|  |
| --- |
| $ bazel build --config=opt --config=cuda //tensorflow/tools/pip\_package:build\_pip\_package |

or:

|  |
| --- |
| $ bazel build --config=opt --config=cuda //tensorflow/tools/pip\_package:build\_pip\_package --action\_env="LD\_LIBRARY\_PATH=${LD\_LIBRARY\_PATH}" --config=monolithic --verbose\_failures |

The bazel build command builds a script named build\_pip\_package. Running this script as follows will build a .whl file within the /tmp/tensorflow\_pkg directory**:**

|  |
| --- |
| $ bazel-bin/tensorflow/tools/pip\_package/build\_pip\_package /tmp/tensorflow\_pkg |

## Install the pip package

Invoke pip install to install that pip package. The filename of the .whl file depends on your platform. For example, the following command will install the pip package

for TensorFlow 1.8.0 on Linux:

|  |
| --- |
| $ pip install /tmp/tensorflow\_pkg/tensorflow....whl |