# Miniconda3, TensorFlow, Keras on Google Compute Engine GPU instance: The step-by-step guide.

vxlabs.com/2017/03/17/miniconda3-tensorflow-keras-on-google-compute-engine-gpu-instance-the-step-by-step-guide/

cpbotha 3/17/2017

Google recently announced the availability of GPUs on Google Compute Engine instances. For my deep learning experiments, I often need more beefy GPUs than the puny GTX 750Ti in my desktop workstation, so this was good news. To make the GCE offering even more attractive, their GPU instances are also available in their EU datacenters, which is in terms of latency a big plus for me here on the Southern tip of the African continent.

Last night I had some time to try this out, and in this post I would like to share with you all the steps I took to:

1. Get a GCE instance with GPU up and running with miniconda, TensorFlow and Keras
2. Create a reusable disk image with all software pre-installed so that I could bring up new instances ready-to-roll at the drop of a hat.
3. Apply the pre-trained Resnet50 deep neural network on images from the web, as a demonstration that the above works. Thanks to Keras, this step is fun and fantastically straight-forward.

## Pre-requisites

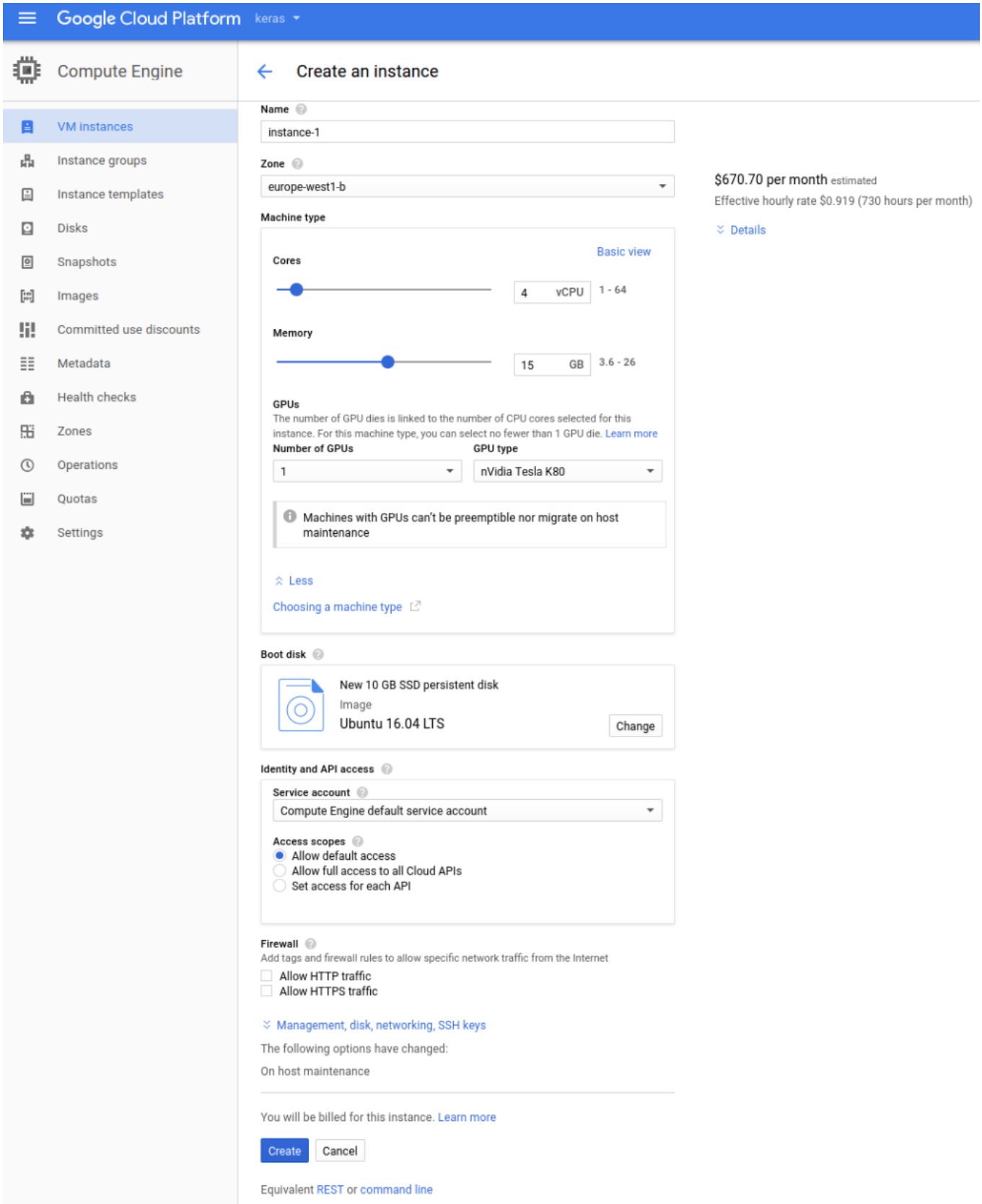
I started by creating a project for this work. On the Compute Engine console, check that this project is active at the top.

Before I was able to allocate GPUs to my instance, I had to fill in the “request quote increase” form available from the Compute Engine quotas page. My request for two GPUs in the EU region was approved within minutes.

I installed my client workstation’s `id_rsa.pub` public SSH key as a project-wide SSH key via the metadata screen.

## Start an instance for the first time

I configured my GPU instance as shown in the following screenshot:




- Under `Machine type` switch to `Customize` to be able to select a GPU.
- I selected an Ubuntu 16.04 image, and changed the persistent disk to SSD.
- I selected the `europe-west1-b` zone. Choose whatever is closest for you. The interface will warn you if the selection does NOT support GPUs.

After this, click on the `Create` button and wait for your instance to become ready.

Once it's up and running, you'll be able to `ssh` to the displayed public IP. I used the ssh on my client workstation, but of course you could opt for the Google-supplied web-based versions.

## Install NVIDIA drivers and CUDA

I used the following handy script from the relevant GCE documentation:

```
#!/bin/bash
echo "Checking for CUDA and installing."
# Check for CUDA and try to install.
if ! dpkg-query -W cuda; then
  # The 16.04 installer works with 16.10.
  curl -O
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-
ubuntu1604_8.0.61-1_amd64.deb
  dpkg -i ./cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
  apt-get update
  apt-get install cuda -y
fi
```

After this, download the CUDNN debs from the NVIDIA download site using your developer account. Install the two debs using `dpkg -i`.

To confirm that the drivers have been installed, run the `nvidia-smi` command:

```
cpbotha@instance-1:~$ sudo /etc/init.d/lightdm stop
[ ok ] Stopping lightdm (via systemctl): lightdm.service.
cpbotha@instance-1:~$ nvidia-smi
Fri Mar 17 11:01:45 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.26                 Driver Version: 375.26                    |
|-------------------------------+----------------------+----------------------+
| 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 K80           Off  | 0000:00:04.0     Off |                    0 |
| N/A   70C    P0    74W / 149W |      0MiB / 11439MiB |     97%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
cpbotha@instance-1:~$
```

## Install miniconda, tensorflow and keras

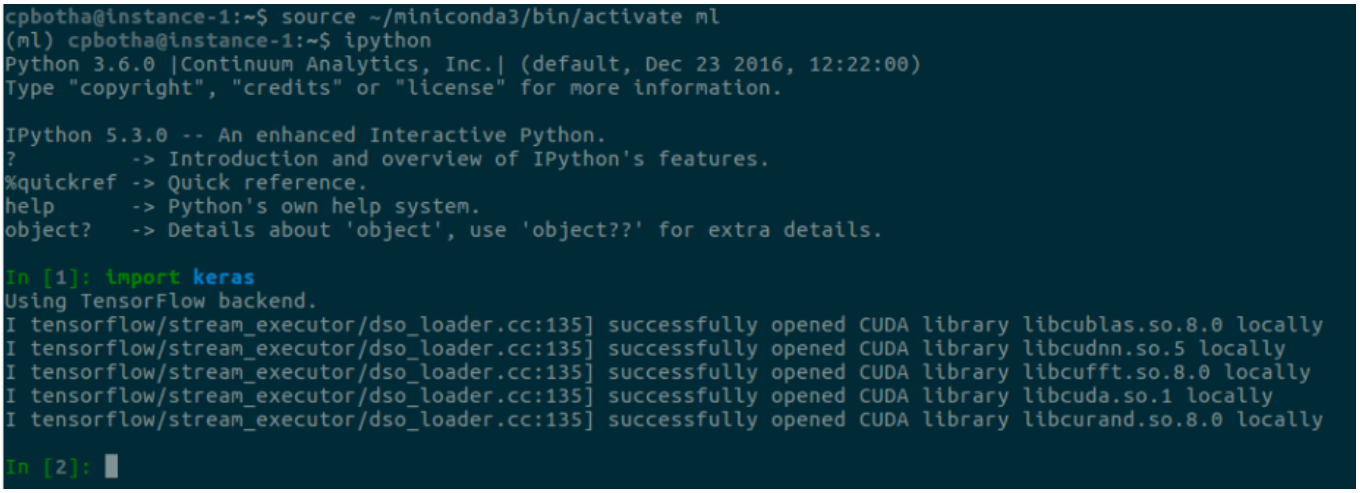
I usually download the 64bit Linux miniconda installer from conda.io and then install it into `~/miniconda3` by running the downloaded `.sh` script.

After this, I installed TensorFlow 1.0.1 and Keras 2.0.1 into a new conda environment by doing:

```
conda create -n ml python=3.6
conda install jupyter pandas numpy scipy scikit-image
pip install --ignore-installed --upgrade
https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.0.1-cp36-cp36m-
linux_x86_64.whl
pip install keras h5py
```

The `keras` package also installed theano, which I then uninstalled using `pip uninstall theano` in the active `ml` environment.

To test, run `ipython` and then type `import keras`. It should look like this:




Note that it's picking up the TensorFlow backend, and successfully loading all of the CUDA librarias, including CUDNN.

## Save your disk as an image for later

You will get billed for each minute that the instance is running. You also get billed for persistent disks that are still around, even if they are not used by any instance.

Creating a reusable disk image will enable you to delete instances and disks, and later to restart an instance with all of your software already installed.

To do this, follow the steps in the documentation, which I paraphrase and extend here:

1. Stop the instance.
2. In the instance list, click on the instance name itself; this will take you to the edit screen.
3. Click the edit button, and then uncheck `Delete boot disk when instance is deleted`.
4. Click the save button.
5. Delete the instance, but double-check that `delete boot disk` is unchecked in the confirmation dialog.
6. Now go to the `Images` screen and select `Create Image` with the boot disk as source.

Next time, go to the `Images` screen, select your image and then select `Create Instance`. That instance will come with all of your goodies ready to go!

## Apply the ResNet50 neural network on images from the interwebs

After connecting to the instance with an SSH port redirect:

```
ssh -L 8889:localhost:8888
cpbotha@EXTERNAL_IP
```

… and then starting a jupyter notebook on the GCE instance:

```
cpbotha@instance-1:~$ source ~/miniconda3/bin/activate
ml
(ml) cpbotha@instance-1:~$ jupyter notebook
```

So that I can connect to the notebook on my `localhost:8889`, I enter and execute the following code (adapted from the Keras documentation) in a cell:

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
from PIL import Image

model = ResNet50(weights='imagenet')

# adapted from https://github.com/fchollet/deep-learning-models
# to accept also a PIL image
def load_and_predict_image(img_or_path):
    target_size = (224,224)
    if type(img_or_path) is str:
        img = image.load_img(img_or_path, target_size=target_size)

    else:
        img = img_or_path.resize(target_size)

    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    preds = model.predict(x)
    # decode the results into a list of tuples (class, description,
probability)
    # (one such list for each sample in the batch)
    print('Predicted:', decode_predictions(preds, top=3)[0])
```

In the next cell, I do:

```
from PIL import Image
import urllib.request

url1 =
"https://upload.wikimedia.org/wikipedia/commons/thumb/c/c8/KuduKr%C3%BCger.jpg/1920px-
KuduKr%C3%BCger.jpg"
url2 = "https://upload.wikimedia.org/wikipedia/commons/thumb/9/9d/Struthio_camelus_-
_Etosha_2014_%283%29.jpg/800px-Struthio_camelus_-_Etosha_2014_%283%29.jpg"
im = Image.open(urllib.request.urlopen(url2))

load_and_predict_image(im)
```

To be greeted with the following results:


The pre-trained ResNet50 network identified the Kudu I gave it initially as an ostrich, so I decided to make it a bit easier for the poor network by actually giving it an ostrich, which it did identify with a 99.98% probability.

Looking at the photos, the former taken in the Kruger National Park and the second in Etosha, I can image that the network could identify the former as the latter due to similar background and foreground colouring, and clearly having not been trained on Kudu.

Let's file that under future work!

***Related***

## Solving the Ubuntu 14.04 - NVIDIA 346 - nvidia-prime black screen issue

February 5, 2015

In "howto"

## Ubuntu 11.04 Natty Narwhal Annoyances (Dell E6410 with NVS 3100m GPU)

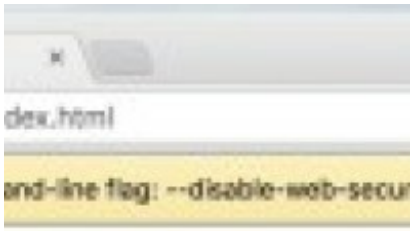
I recently upgraded my Dell E6410 with NVS 3100m GPU laptop from Ubuntu 10.10 (Maverick Meerkat) to 11.04

(Natty Narwhal), and I can't shake this feeling that the distribution has taken a few steps back. I'm not even referring to the new Unity desktop, but to some super-irritating annoyances I…

May 15, 2011

In "all"

## Fixing the Cordova browser platform Access-Control-Allow-Origin error




March 17, 2016

In "howto"