AI Practitioners Guide for Beginners

TensorFlow* Framework Deployment and Example Test Runs on Intel® Xeon® Platform-based Infrastructure

Scope

This practitioner enablement guide provides a high-level overview of business and data strategy that a machine learning (ML) practitioner needs to know, followed by a detailed walkthrough of how to install and validate one of the popular artificial intelligence (AI) frameworks, TensorFlow* on Intel® Xeon® Scalable platforms. The guide details steps for installing running popular examples three different ways: on bare metal, via containers, and on the cloud.

Note: the examples shown in this guide were not performance optimized and are for educational purpose only.

Contents

Definition of Artificial Intelligence (AI)

The definition of "artificial intelligence" is continually evolving, but at its core, AI is about machines mimicking (and/or exceeding) cognitive functions associated with the human mind. In the universe of AI, which includes many different approaches, data-centric machine learning has emerged as a leader due to its increasing ability to tackle the three main AI sub-tasks: perception, planning/reasoning, and control. Ultimately, AI is achieved through the fusion of multiple approaches to deliver ever more intelligent machines, and the nexus of AI developments in the near-future is centered on deep learning, with other approaches all playing important roles – depending on the dataset, problem, and unique requirements.

As shown in figure 1, a subset of AI umbrella is machine learning, which can be defined as machine algorithms whose performance keeps improving as they are exposed to more data over time. For example, if you were to program a self-learning robot to water your plants in the garden and that robot hits a stone on its way, it will learn to avoid the obstacle and take the optimized path in the future. Hence, the garden robot's machine learning helps improve its performance over time.

A subset of machine learning is deep learning (DL), where multi-layered neural networks learn from vast amounts of data. Deep learning is the branch of AI that has gained huge popularity and adoption in recent years. The framework and examples provided in this guide are based on deep learning. DL comprises two major pieces, training and inference. Training teaches multi-layered neural networks (also known as models) to identify objects/text, etc. by feeding labeled data/content into it. Once the model is trained, inference begins, using the trained model to identify unlabeled content.

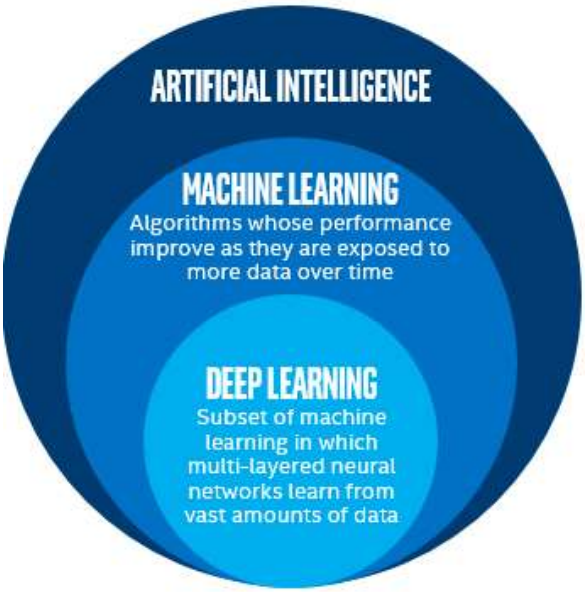


Figure 1: AI and its major subsets

Business Considerations

Determine a Data Strategy[1]

The business imperative for AI is firmly rooted in data – the currency of the future. By 2020, we expect over 50 billion devices and 200 billion sensors to join the internet, and this huge explosion of smart and connected devices will lead to incalculable volumes of data being generated. In 2020, it is expected that the **average internet user** will generate **~1.5 GB of traffic per day** (up from ~650MB in 2015). This is certainly a huge amount of data … until you consider the machines:

- A smart hospital will generate 3,000 GB/day
- Self-driving cars are each generating over 4,000 GB/day
- A connected plane will generate 5,000 gigabytes per day
- A connected factory will generate 1 million gigabytes per day

This data contains extremely valuable insights for business, operations, and security that affected industries really want to extract, analyze and interpret in real time. Extracting value from that data requires all the AI tools at our disposal.

The first step on your AI journey is to prepare your **data**. And for that, when thinking of an AI business model, it's imperative to focus on the entire data lifecycle.

Figure 2: Added AI data life cycle

Your data strategy should include a clear plan for how to create, source, transmit, ingest, clean, and integrate diverse data, followed by how to store and stage the data before processing. Each organization will have a unique data strategy but remember the data lifecycle and focus on building an end-to-end optimized data-based solution to arrive at a unique, competitive data strategy.

Analyze the Business Problem You are Trying to Solve[1]

Before exploring AI, it is important to understand that implementing AI in your organization will be a journey. The first step is to define the challenges you're facing across your organization and prioritizing them based on business value and how much it will cost to solve them. Picture a 2x2 chart with increasing business value on the y-axis and decreasing the cost to solve on the x-axis; naturally, the most impactful challenges to tackle first are in the upper-right quadrant. The next steps are to determine which AI (or other) approach is best-suited to each problem, and then assess whether you have the expertise required to implement the solution. (Additionally, you should know whether those experts embrace a fail-fast continuous improvement philosophy, since AI projects typically involve more uncertainty, trial and error, and exploration than more traditional and deterministic software

development projects.) Once the human element is in place, the next step is to source data and prepare it for analysis, as well as to stand up whatever technology infrastructure is required to tackle the problem.

Finally, you're ready to do the heavy lifting to use data to solve business challenges – but unless your organization is ready to accept and act on data-driven insights, then all that work may have been for naught. A classic example is an initial resistance to data analytics in sports, where general managers and scouts scoffed at the idea of computer algorithms outsmarting their years of experience and tribal knowledge. Bottom line: if you think about all these steps in the AI lifecycle, you'll stand a much better chance of realizing the business value that you originally set out to deliver through AI.

Professor Thomas Malone from MIT Sloan School of Management is founding director of the MIT Center for Collective Intelligence. In several of his works, he explores the idea of humans and machines working collectively to change the world. Understanding how this collective intelligence of humans and AI can affect your business strategy is critical. Combining a strong data strategy with a deep understanding of the business problem you're trying to solve with AI will help you accelerate your business in the future.

TensorFlow* Framework Deployment and Examples

This section details on the three ways to deploy a TensorFlow framework for deep learning training and inference for an Intel Xeon platform-based infrastructure.

	Single Node	**Multi-node**
Option 1	Bare metal	Bare metal
Option 2	Via containers	X (Not covered in this document. Please check references [6, 29])
Option 3	On Cloud	On Cloud

Option 1: Bare Metal

Single Node Installation:

This section details how to train and test a single-node Intel® Xeon® Scalable processor system using a TensorFlow framework with CIFAR-10 image recognition datasets. Use these step-by-step instructions as-is, or as the foundation for enhancements and/or modifications.

Knowledge Prerequisites:

Hardware	Steps have been verified on Intel® Xeon® Scalable processors but should work on any latest Intel Xeon processor-based system. None of the software pieces used in this document were performance optimized.
Software	Basic Linux* and familiarity with the concepts of deep learning training

This section describes one way to successfully deploy and test an image recognition example on a single Intel Xeon Scalable processor system running CentOS* 7.3. Other installation methods can be found in Installing TensorFlow* on Ubuntu*, Intel® Optimization for TensorFlow* Installation Guide[3]. This document used a virtual environment for installing TensorFlow. To use Anaconda*, first, refer to this article. This document is not meant to describe how to achieve state-of-the-art performance; rather, it's to introduce TensorFlow and run a simple train and test using examples like the CIFAR-10 dataset on various Intel Xeon processor-based systems.

Hardware and Software Bill of Materials

Item	Manufacturer	Model/Version
Hardware		
Intel-based server chassis	Intel	R1208WT
Intel-based server board	Intel	S2600WT
(2x) Intel® Xeon® Scalable processor	Intel	Intel® Xeon® Gold 6148 processor
(6x) 32GB LRDIMM DDR4	Crucial*	CT32G4LFD4266
(1x) Intel® SSD 1.2TB	Intel	S3520
Software		
CentOS* Linux* Installation DVD		7.3.1611
Intel® Parallel Studio XE Cluster Edition		2017.4
TensorFlow*		setuptools-36.7.2-py2.py3-none-any.whl

Step 1: Install the Linux* Operating System

In this section, CentOS7.3.1611 was used. Download an updated version of the software from the CentOS website.

Find steps for OS installation in the Appendix.

Step 2. Configure YUM

If the public network implements a proxy server for internet access, Yellowdog Updater Modified* (YUM*) must be configured in order to use it.

Open the /etc/yum.conf file for editing.

Under the main section, append the following line:
proxy=http://<address>:<port>;
where <address> is the address of the proxy server and <port> is the HTTP port.

Save the file and Exit.

Disable updates and extras. Certain procedures in this document require packages to be built against the kernel. A future kernel update may break the compatibility of these built packages with the new kernel, so disabling repository updates and extras is recommended to provide further longevity to this document.

This document may not be used "as is" when CentOS updates to the next version. To use this document after such an update, it is necessary to redefine repository paths to point to CentOS 7.3 in the CentOS vault. To disable repository updates and extras: Yum-config-manager --disable updates --disable extras.

Step 3. Install EPEL

Extra Packages for Enterprise Linux (EPEL) provides 100 percent, high-quality add-on software packages for Linux distribution. To install EPEL (latest version for all packages required):

Yum –y install (download from here)

Step 4. Install GNU* C Compiler

Check whether the GNU Compiler Collection* (GCC*) is installed. It should be part of the Development Tools install in OS installation. (Look in the Appendix.) Check by typing:

gcc --version or whereis gcc

If not installed, find the latest installation here.

GCC can be installed from the official CentOS repository by using the following command:

yum –y install gcc

Step 5. Install TensorFlow*

Using virtualenv[3], follow these steps to install TensorFlow:

1. Update to the latest distribution of EPEL:

```
yum –y install epel-release
```

2. To install TensorFlow, the following dependencies must be installed:

- NumPy*: a numerical processing package that TensorFlow requires
- Devel*: enables adding extensions to Python*
- PIP*: enables installing and managing certain Python packages
- Wheel*: enables managing Python compressed packages in wheel formal (.whl)
- Atlas*: Automatically Tuned Linear Algebra Software
- Libffi*: Library provides Foreign Function Interface (FFI) that allows code written in one language to call code written in another language. It provides a portable, high-level programming interface to various calling conventions[8]

3. Install dependencies:

```
sudo yum -y install gcc gcc-c++ python-pip python-devel atlas
atlas-devel gcc-gfortran openssl-devel libffi-devel python-numpy
```

4. Install virtualenv

There are various ways to install TensorFlow. This document uses virtualenv, a tool to create isolated Python environments[9].

```
pip install --upgrade virtualenv
```

5. Create a virtualenv in your target directory:

```
virtualenv --system-site-packages <targetDirectory>
```
Example: `virtualenv --system-site-packages tensorflow`

6. Activate your virtualenv[4]:

```
source ~/<targetDirectory>/bin/activate
```
Example: `source ~/tensorflow/bin/activate`

7. Upgrade your packages, if needed:

```
pip install --upgrade numpy scipy wheel cryptography
```

8. Install the latest version of Python compressed TensorFlow packages:

```
Pip install --upgrade
```

This document was deployed and tested using TensorFlow 0.8 wheel.

Google releases an updated version of TensorFlow on a regular cadence, so using the latest available version of TensorFlow wheel is recommended.

Find the latest version of Intel MKL-DNN optimized Tensor wheel file in GitHub*, under Community Supported Builds.

Example[4]:

Linux CPU with Intel® MKL-DNN Python 2.7		1.9.0 py2.7
Linux CPU with Intel® MKL-DNN Python 3.5		1.9.0 py3.5
Linux CPU with Intel® MKL-DNN Python 3.6		1.9.0 py3.6

CPU optimized TensorFlow 1.9 wheel[19] file can be downloaded as follows:

> Python* 2.7: `pip install https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide#pip_wheels'`
>
> Python* 3.5: `pip install https://storage.googleapis.com/intel-optimized-tensorflow/tensorflow-1.9.0-cp35-cp35m-linux_x86_64.whl`

Versions of CPU-only wheel files are available on TensorFlow webpage, which can also be used. However, these may not be optimized for CPUs.

After installing a version of TensorFlow wheel, you have the option to upgrade to the latest TensorFlow, but be advised that the upgraded version might not be CPU optimized.

Step 6. Train a Convolutional Neural Network (CNN)

1. Download the CIFAR10[11] training dataset into `/tmp/` directory, the Python version can be found here.

2. Unzip the tar file in the /tmp/ area as the Python script (cifar10_train.py) looks for data in this directory:

```
tar -zxf <dir>/cifar-10-python.tar.gz
```

3. Change directory to TensorFlow:

```
cd tensorflow
```

4. Make a new directory:

```
mkdir git_tensorflow
```

5. Change directory to the one created in the last step:

```
cd git_tensorflow
```

6. Download a clone of the TensorFlow repository from GitHub: Git clone

7. If the Models folder is missing from the tensorflow/tensorflow directory, access a Git of models from TensorFlow Github[13]:

```
cd tensorflow/tensorflow
git clone https://github.com/tensorflow/models.git
```

8. Upgrade TensorFlow to the latest version or errors could occur when training the model:

```
pip install --upgrade tensorflow
```

9. Change directory to CIFAR-10 dir to get the training and evaluation Python scripts[12]:

```
cd models/tutorials/image/cifar10
```

10. Before running the training code, check the cifar10_train.py code and change steps from 100K to 60K if needed, as well as logging frequency from 10 to whatever you prefer.

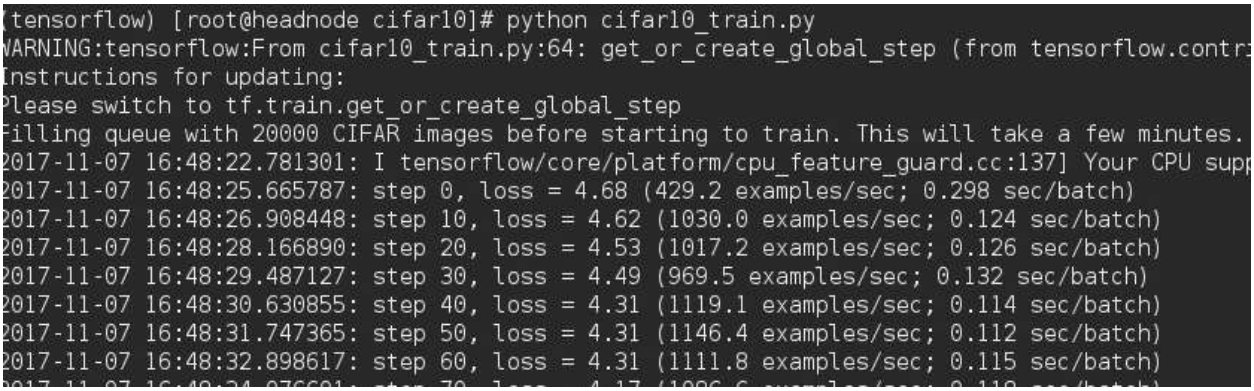
For this document, tests were done for both 100K steps and 60K steps, for a batch size of 128, and logging frequency of 10.

```
parser.add_argument('--max_steps', type=int, default=100000,
help='Number of batches to run.')
```

11. Run the training Python script to train your network:

```
python cifar10_train.py
```

This will take a few minutes and you will see an image similar to below:



Testing Script and Dataset Terminology

In the neural network terminology:

- One **epoch** = one forward pass and one backward pass of all the training examples.
- **Batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space required. TensorFlow pushes it all through one forward pass (in parallel) and follows with a back-propagation on the same set. This is one iteration or step.
- Number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass equals one forward pass plus one backward pass (do not count the forward pass and backward pass as two different passes).
- **Steps** parameter tells TensorFlow to run X of these iterations to train the model.
  Example: given 1,000 training examples, and a batch size of 500, it will take two iterations to complete one epoch.

To learn more about the differences between epoch, batch size, and iterations, read the Performance Guide for TensorFlow.

In the cifar10_train.py script:

- Batch size is set to 128. It represents the number of images to process in a batch.
- Max step is set to 100,000. It is the number of iterations for all epochs.

Note: The GitHub code has a typo; instead of 100K, the number shows 1000K. Please update before running.

- The CIFAR-10 binary dataset in Intel® Optimization for TensorFlow* Installation Guide[4] has 60,000 images: 50,000 images to train and 10,000 images to test. Each batch size is 128, so the number of batches needed to train is 50,000/128 ~ 391 batches for one epoch.
- The cifar10_train.py used 256 epochs, so the number of iterations for all the epochs is ~391 x 256 ~ 100K iterations or steps.

Step 7. Evaluate the Model

Use the cifar10_eval.py script[8] to evaluate how well the trained model performs on a hold-out dataset:

```
python cifar10_eval.py
```

Once you reach expected accuracy, you should see a precision @ 1 = 0.862 onscreen when running the above command. It can be run while the training script is still running toward the end of the number of steps, or it can be run after the training script has finished.

```
2017-11-13 20:08:20.976879: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports
 instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 AVX512F
FMA
2017-11-13 20:08:23.281557: precision @ 1 = 0.862
```

A similar-looking result below was achieved with the system described in the Hardware and Software Bill of Materials Section of this document.

Note that these numbers are only for educational purposes and no specific CPU optimizations were performed.

System	Step Time (sec/batch)	Accuracy
2 - Intel® Xeon® Gold 6148 processor	~ 0.105	85.8% at 60K steps (~2 hours)
2 - Intel® Xeon® Gold 6148 processor	~0.109	86.2% at 100K steps (~3 hours)

When you finish training and testing your CIFAR-10 dataset, the same Models directory has images for MNIST* and AlexNet benchmarks. For additional learning, go into MNIST and AlexNet directories and try running the Python scripts to see the results.

Multiple Node Installation:

Wei Wang and Mahmoud Abuzaina in their article have provided details on achieving performance scaling using Intel Xeon Scalable processors and Horovod* with TensorFlow. Their blog has been the source of the content for this section.

Many complex deep learning models are required to be trained on multi-node. This is because they either don't fit in one machine or their time-to-train can be significantly reduced if they are trained on a cluster of machines. Therefore, Intel has also performed scaling studies on multi-node clusters of Intel Xeon Scalable processors. This section will provide steps to deploy TensorFlow on clusters of Intel Xeon processors using Horovod, a distributed training framework for TensorFlow.

Horovod, which was developed by Uber*, uses a message passing interface (MPI) as the main mechanism of communication. It uses MPI concepts such as allgather and allreduce to handle the cross-replicas communication and weight updates. OpenMPI* can be used with Horovod to support these concepts. Horovod is installed as a separate Python package. By calling Horovod's API from the deep learning neural networks model script, a regular build of TensorFlow can be used to run distributed training. By using Horovod, there is no source code change required in TensorFlow to support distributed training with MPI.

Hardware and Software Bill of Materials

Item	Manufacturer	Model/Version
Hardware		
Intel® Xeon® Scalable processor	Intel	Intel® Xeon® Gold 6148 processor
(12x) 16GB DDR4 @ 2666MT/s		
(3x) Intel® SSD 800GB, 1.6TB	Intel	RS3WC080
Software		
CentOS*		CentOS 7.4 (Maipo)
Kernel		3.10.0-693.21.1.0.1.el7.knl1.x86_64
TensorFlow*		1.7

Step 1: Install the Linux* Operating System

In this section CentOS* 7.4 was used. Download an updated version of the software from the CentOS website.

Find steps for OS installation in the Appendix.

This white paper assumes that a multiple node cluster has been set up and there is communication between the head node and compute nodes. Refer to HPC Cluster Reference Design[19] if guidance is needed for cluster setup.

Step 2. Configure YUM*

If the public network implements a proxy server for internet access, Yellowdog Updater Modified* (YUM*) must be configured in order to use it.

Open the `/etc/yum.conf` file for editing.

Under the main section, append the following line:

```
proxy=http://<address>:<port>;
```

where `<address>` is the address of the proxy server and `<port>` is the HTTP port.

`Save` the file and **`exit`**.

Disable updates and extras. Certain procedures in this document require packages to be built against the kernel. A future kernel update may break the compatibility of these built packages with the new kernel, so disabling repository updates and extras is recommended to provide further longevity to this document.

This document may not be used "as is" when CentOS updates to the next version. To use this document after such an update, it is necessary to redefine repository paths to point to CentOS 7.4 in the CentOS vault. To disable repository updates and extras: Yum-config-manager --disable updates --disable extras.

Step 3. Install EPEL

Extra Packages for Enterprise Linux (EPEL) provides 100 percent, high-quality add-on software packages for Linux distribution. To install EPEL (latest version for all packages required):

Download:

```
yum –y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Step 4. Install GNU* C Compiler

Check whether the GNU Compiler Collection* (GCC*) is installed. Should be part of the Development Tools install in OS installation (Check Appendix). You can check by typing:

```
gcc --version or whereis gcc
```

If not installed, find the latest installation here.

GCC can be installed from the official CentOS* repository by using the following command:

```
yum –y install gcc
```

Step 4. Install OpenMPI

OpenMPI can be installed via yum on recent versions of CentOS. Some existing clusters already have available OpenMPI. In this section, we will use OpenMPI 3.0.0. OpenMPI can be installed following instructions in this link.

Example installation of rpm file after download:

```
yum localinstall openmpi-3.0.0-1.src.rpm
```

Step 5. Python installation

Make sure Python* 2.7 or Python* 3.6 are installed and tested. As part of the OS installation, necessary packages must have been installed. Update all necessary packages as follows:

```
Sudo yum update
Sudo yum install yum-utils
Sudo yum groupinstall development
```

Proceed with installing Python. This section provides steps to install Python 3.6.1. The standard yum repositories don't provide the latest Python release, so an additional repository in-line with upstream stable (IUM) is needed, as it provides necessary RPM packages.

```
sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
sudo yum install python36u
```

Check the version of Python 3 by typing:

```
python3.6 –V
```

`python –V` will return the system Python version

To manage Python packages, install pip and needed development packages, if not already there.

```
sudo yum install python36u-pip
sudo yum install python36u-devel
```

Step 6. Horovod* installation

Uber Horovod supports running TensorFlow in a distributed fashion. Install Horovod as a standalone Python package as follows:

```
pip install –no-cache-dir horovod (e.g. horovod-0.11.3)
```

Please check the following link to install Horovod from this source:

Step 7. Get the Latest benchmarks

The current TensorFlow benchmarks are recently modified to use Horovod. Obtain the benchmark code from GitHub:

```
git clone https://github.com/tensorflow/benchmarks
cd benchmarks/scripts/tf_cnn_benchmarks
```

Run `tf_cnn_benchmarks.py` as explained below.

Step 8: Running TensorFlow* benchmark using Horovod*

This section discusses run commands needed to run distributed TensorFlow using Horovod framework.

Running 2 MPI processes on single node:

```
export LD_LIBRARY_PATH=<path to OpenMP lib>:$LD_LIBRARY_PATH
export PATH=<path to OpenMPI bin>:$PATH
export inter_op=2
export intra_op=18 {# cores per socket}
export batch_size=64
export MODEL=resnet50 {or inception3}
export python_script= {path for tf_cnn_benchmark.py script}

mpirun -x LD_LIBRARY_PATH -x OMP_NUM_THREADS -cpus-per-proc 20 --map-
by socket  --overscribe
--report-bindings -n 2 python $python_script --mkl=True --
forward_only=False --num_batches=200
--kmp_blocktime=0 --num_warmup_batches=50 --
num_inter_threads=$inter_op --distortions=False
--optimizer=sgd --batch_size=$batch_size --num_intra_threads=$intra_op
--data_format=NCHW
--model=$MODEL --variable_update horovod --horovod_device cpu --
data_dir <path-to-real-dataset>
--data_name <dataset_name>
```

For 1 MPI process per node, the configuration will be as follows; other environment variables will remain the same.

```
export intra_op=38
export batch_size=128

mpirun -x LD_LIBRARY_PATH -x OMP_NUM_THREADS --bind-to none --report-
bindings
-n 1 python $python_script --mkl=True --forward_only=False --
num_batches=200
--kmp_blocktime=0 --num_warmup_batches=50 --
num_inter_threads=$inter_op
--distortions=False --optimizer=sgd --batch_size=$batch_size
--num_intra_threads=$intra_op --data_format=NCHW --model=$MODEL
--variable_update horovod --horovod_device cpu --data_dir <path-to-
real-dataset>
--data_name <dataset_name>
```

Note: to train models to achieve good accuracy, use –distortions=True. You may also need to change other hyper-parameters.

For running models on a multi-node cluster, use a similar run script as the one above. For example, to run on 64-node (2 MPI per node), where each node is an Intel Xeon Gold 6148 processor, the distributed training can be launched as shown below. All the export lists will be the same as above.

```
mpirun -x LD_LIBRARY_PATH -x OMP_NUM_THREADS -cpus-per-proc 20 --map-
by node
--report-bindings -hostfile host_names -n 128 python $python_script --
mkl=True
--forward_only=False --num_batches=200 --kmp_blocktime=0 --
num_warmup_batches=50
--num_inter_threads=$inter_op --distortions=False --optimizer=sgd --
batch_size=$batch_size
 --num_intra_threads=$intra_op --data_format=NCHW --model=$MODEL --
variable_update horovod
--horovod_device cpu --data_dir <path-to-real-dataset> --data_name
<dataset_name>
```

Here, the `host_names` file is the list of hosts on which you wish to run the workload.

What Distributed TensorFlow* Means for DL Training on Intel® Xeon® Processors

Various efforts were taken to implement distributed TensorFlow on CPU and GPU. For example, gRPC, VERBS, TensorFlow built-in MPI. All of these technologies are incorporated within TensorFlow codebase. Uber Horovod is one distributed TensorFlow technology that was able to harness the power of Intel Xeon processors. It uses MPI underneath, and uses Ring based reduction and gather for deep learning parameters. As shown in the blog by Wang and Abuzaina, Horovod on Intel Xeon processors shows great scaling for existing DL benchmark models, such as Resnet 50 (up to 94%) and Inception v3 (up to 89%) for 64 nodes. In other words, time to train a DL network can be accelerated by as much as 57x (resnet 50) and 58x (inception V3) using 64 Intel Xeon processor nodes compared to a single such node. Currently, Intel recommends that TensorFlow users specify Intel-optimized TensorFlow and Horovod MPI for multi-node training on Intel Xeon Scalable processors.

Option 2: Using AI Containers

Single Node Installation:

Hardware and Software Bill of Materials

Item	Manufacturer	Model/Version
Hardware		
(2x) Intel® Xeon® Scalable processor	Intel	Intel® Xeon® Platinum 8164 processor
(12x) 32GB DDR4 @ 2666MT/s		
(3x) Intel® SSD 800GB, 1.6TB	Intel	RS3WC080

Software		
CentOS		CentOS 7.5
Kernel		3.10.0-862.el7.x86_64
TensorFlow*		1.9

Step 1: Install the Linux* Operating System

In this section, CentOS*7.4 was used. Download an updated version of the software from the CentOS website.

Find steps for OS installation in the Appendix.

Step 2. Configure YUM

If the public network implements a proxy server for internet access, Yellowdog Updater Modified* (YUM*) must be configured in order to use it.

Open the `/etc/yum.conf` file for editing.

Under the main section, append the following line:
`proxy=http://<address>:<port>;`
where `<address>` is the address of the proxy server and `<port>` is the HTTP port.

`Save` the file and `Exit`.

Disable updates and extras. Certain procedures in this document require packages to be built against the kernel. A future kernel update may break the compatibility of these built packages with the new kernel, so disabling repository updates and extras is recommended to provide further longevity to this document.

This document may not be used "as is" when CentOS updates to the next version. To use this document after such an update, it is necessary to redefine repository paths to point to CentOS 7.4 in the CentOS vault. To disable repository updates and extras:

```
Yum-config-manager --disable updates --disable extras.
```

Step 3. Install EPEL

Extra Packages for Enterprise Linux (EPEL) provides 100 percent, high-quality add-on software packages for Linux distribution. To install EPEL (latest version for all packages required):

`yum –y install` (download from here)

Step 4. Install GNU* C Compiler

Check whether the GNU Compiler Collection* (GCC*) is installed. Should be part of the Development Tools install in OS installation (Check Appendix). You can check by typing:

```
gcc --version or wheeis gcc
```

If not installed, find the latest installation here.

Install GCC from the official CentOS* repository by using the following command:

```
yum –y install gcc
```

Step 5. Download and Install Anaconda

Follow the instructions on the Anaconda download site to download and install Anaconda.

Download the source file for Anaconda for Python* 2.7

(Python* 2.7 is recommended as currently TensorFlow is only supported for Python* 2.7 or Python* 3.5. This section uses Python* 2.7)

Install Anaconda by using following command

```
bash Anaconda-latest-Linux-x86_64.sh
```

Follow the prompts on the screen to complete the installation.

Note: You will need to open a new terminal to for the Anaconda installation to become active.

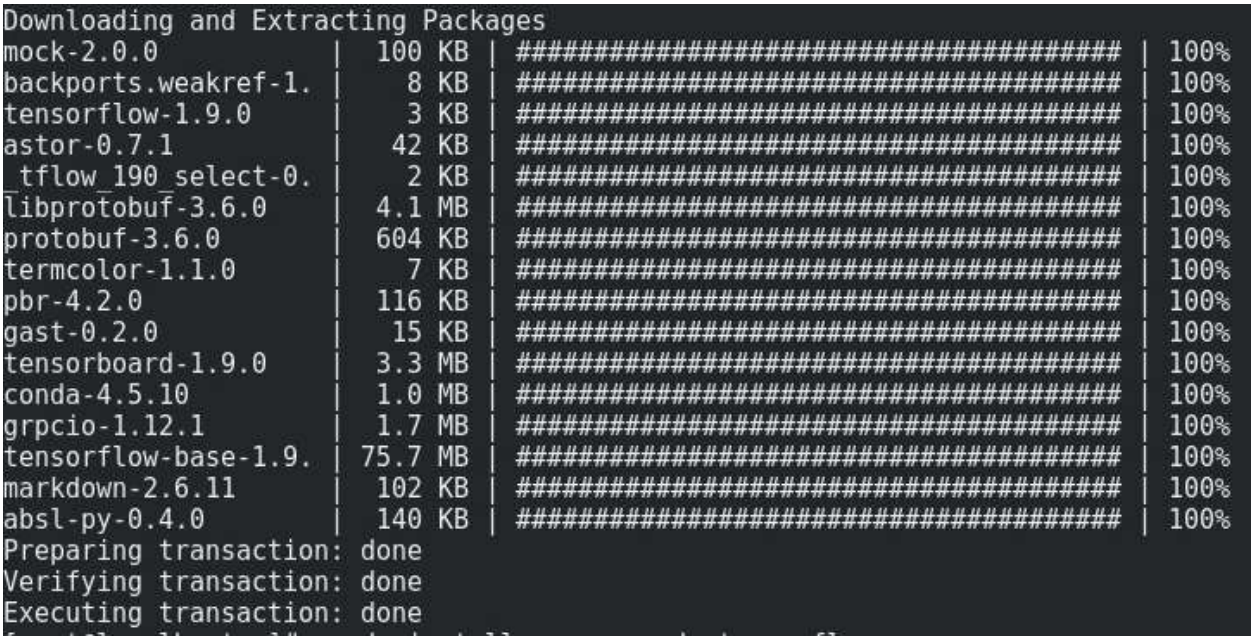
Step 6. Install the Latest Intel® Optimization for TensorFlow* from Anaconda

Open the Anaconda prompt using the following instruction:

```
conda install tensorflow
```

Follow the prompts onscreen to complete downloading and extracting the packages.

Expect to see a screen similar to the following:



If your anaconda channel is not the highest priority channel by default (or if you are unsure), use the following command to get the correct Intel® Optimization for TensorFlow*:

```
conda install –c anaconda tensorflow
```

Expect to see a screen similar to the following:

```
[root@localhost ~]# conda install -c anaconda tensorflow
Solving environment: done

## Package Plan ##

  environment location: /root/anaconda2

  added / updated specs:
    - tensorflow


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.5.10               |           py27_0         1.0 MB  anaconda
    tensorflow-1.9.0           |mkl_py27h0cb61a4_1           3 KB  anaconda
    certifi-2018.4.16          |           py27_0         142 KB  anaconda
    openssl-1.0.2o             |       h20670df_0         3.4 MB  anaconda
    ca-certificates-2018.03.07 |                0         124 KB  anaconda
    ------------------------------------------------------------
                                           Total:         4.7 MB

The following packages will be UPDATED:

    ca-certificates: 2018.03.07-0             --> 2018.03.07-0             anaconda
    certifi:         2018.4.16-py27_0         --> 2018.4.16-py27_0         anaconda
    conda:           4.5.10-py27_0            --> 4.5.10-py27_0            anaconda
    openssl:         1.0.2o-h20670df_0        --> 1.0.2o-h20670df_0        anaconda
    tensorflow:      1.9.0-mkl_py27h0cb61a4_1 --> 1.9.0-mkl_py27h0cb61a4_1 anaconda

Proceed ([y]/n)? y


Downloading and Extracting Packages
conda-4.5.10         | 1.0 MB    | ##################################### | 100%
tensorflow-1.9.0     | 3 KB      | ##################################### | 100%
certifi-2018.4.16    | 142 KB    | ##################################### | 100%
openssl-1.0.2o       | 3.4 MB    | ##################################### | 100%
ca-certificates-2018 | 124 KB    | ##################################### | 100%
```

Besides the install method described above, Intel Optimization for TensorFlow is distributed as wheels, docker images and conda package on the web page Intel channel. This section will cover installing Intel Optimization for TensorFlow using docker images.

Step 7. Install Docker

Install Docker on your system; or, skip to step 8 if Docker is already installed.

Install Docker on CentOS

```
yum install docker
```

Once complete, expect to see a screen similar to the following:

```
Installed:
  docker.x86_64 2:1.13.1-74.git6e3bb8e.el7.centos

Dependency Installed:
  atomic-registries.x86_64 1:1.22.1-22.git5a342e3.el7                container-selinux.noarch 2:2.68-1.el7
  container-storage-setup.noarch 0:0.11.0-2.git5eaf76c.el7           docker-client.x86_64 2:1.13.1-74.git6e3bb8e.el7.centos
  docker-common.x86_64 2:1.13.1-74.git6e3bb8e.el7.centos             oci-register-machine.x86_64 1:0-6.git2b44233.el7
  oci-systemd-hook.x86_64 1:0.1.17-2.git83283a0.el7                  oci-umount.x86_64 2:2.3.3-3.gite3c9055.el7
  python-pytoml.noarch 0:0.1.14-1.git7dea353.el7                     skopeo-containers.x86_64 1:0.1.31-1.dev.gitae64ff7.el7.centos
  subscription-manager-rhsm-certificates.x86_64 0:1.20.11-1.el7.centos

Complete!
```

Install epel repositories, which must be enabled on your system.

```
yum install epel-release
```

```
yum install docker-io
```

After the Docker package has been installed, start the daemon. Enable it system-wide and check its status by using the following commands:

```
systemctl start docker
```

```
systemctl status docker
```

```
systemctl enable docker
```

```
[root@localhost ~]# systemctl start docker
[root@localhost ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2018-08-22 10:45:04 PDT; 13s ago
     Docs: http://docs.docker.com
 Main PID: 120686 (dockerd-current)
    Tasks: 49
   CGroup: /system.slice/docker.service
           ├─120686 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-run
           └─120696 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metr

Aug 22 10:45:02 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:02.277332795-07:00" level=info msg="li
Aug 22 10:45:03 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:03.399901667-07:00" level=info msg="Gr
Aug 22 10:45:03 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:03.401984199-07:00" level=info msg="Lo
Aug 22 10:45:03 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:03.431202696-07:00" level=info msg="Fi
Aug 22 10:45:03 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:03.740551122-07:00" level=info msg="De
Aug 22 10:45:04 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:04.017717305-07:00" level=info msg="Lo
Aug 22 10:45:04 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:04.047686432-07:00" level=info msg="Da
Aug 22 10:45:04 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:04.047734450-07:00" level=info msg="Do
Aug 22 10:45:04 localhost.localdomain dockerd-current[120686]: time="2018-08-22T10:45:04.059711175-07:00" level=info msg="AP
Aug 22 10:45:04 localhost.localdomain systemd[1]: Started Docker Application Container Engine.
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
[root@localhost ~]#
```

Finally, run a container test image to verify that Docker works properly, using the following command:

```
docker run hello-world
```

If Docker is working properly, expect to see something like the following:

```
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for docker.io/hello-world:latest

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

Note: In case of issues with Docker connection timeout, and you are behind a proxy server (for example in a corporate setting), you may need to add certain configurations in the *Docker system service file*.

Step 8. Install the Latest Intel® Optimization for TensorFlow* Docker Images Into an Existing Python* Installation

These Docker images are all published at dockerhub in the intelaipg/intel-optimized-tensorFlow namespace and can be pulled with the following command:

```
docker pull docker.io/intelaipg/intel-optimized-tensorflow:<tag>
```

Example:

```
docker pull docker.io/intelaipg/intel-optimized-
tensorflow:latest-devel-mkl
```

Available container configurations and tags can be found here.

Once the Docker pull is complete, expect to see a screen similar to the following:

```
[root@localhost TensorFlow_container]# docker pull docker.io/intelaipg/intel-optimized-tensorflow:latest-devel-mkl
Trying to pull repository docker.io/intelaipg/intel-optimized-tensorflow ...
latest-devel-mkl: Pulling from docker.io/intelaipg/intel-optimized-tensorflow
b234f539f7a1: Pull complete
55172d420b43: Pull complete
5ba5bbeb6b91: Pull complete
43ae2841ad7a: Pull complete
f6c9c6de4190: Pull complete
1cf6d06b5ba7: Pull complete
ebee93cf329b: Pull complete
13288e86382e: Pull complete
65fb99a7fb6b: Pull complete
a6060a576d21: Pull complete
d106c722c2f0: Pull complete
8211e6574afe: Pull complete
182336a8b301: Pull complete
4ef1729a4083: Pull complete
aff3d1a76ce6: Pull complete
0c121428cc24: Pull complete
78b9d6feff90: Pull complete
7419f8375b9c: Pull complete
0cdb7ac55670: Pull complete
1c8aaeac0c47: Pull complete
3d7bb1f9b309: Pull complete
Digest: sha256:747bb58eba4a3773d2d0c481bb84a8b6c49e82809db722fcae67f4d650092b06
Status: Downloaded newer image for docker.io/intelaipg/intel-optimized-tensorflow:latest-devel-mkl
[root@localhost TensorFlow_container]#
```

To see the list of all available Docker images on your system, type following command:

```
docker images
```

```
[root@localhost TensorFlow_container]# docker images
REPOSITORY                                    TAG                 IMAGE ID            CREATED             SIZE
docker.io/intelaipg/intel-optimized-tensorflow   latest-devel-mkl    bf3f3bb7a330        3 weeks ago         2.98 GB
docker.io/intelaipg/intel-optimized-tensorflow   latest              b12e006dca23        3 weeks ago         1.91 GB
docker.io/hello-world                         latest              2cb0d9787c4d        6 weeks ago         1.85 kB
```

Now you can run an example of Python* 2.7 data science container and have it open in Jupyter* Notebook by typing:

```
docker run -it -p 8888:8888 intelaipg/intel-optimized-tensorflow
```

Go to your browser on http://localhost:8888/

```
[root@localhost TensorFlow_container]# docker run -it -p 8888:8888 intelaipg/intel-optimized-tensorflow
Unable to find image 'intelaipg/intel-optimized-tensorflow:latest' locally
Trying to pull repository docker.io/intelaipg/intel-optimized-tensorflow ...
latest: Pulling from docker.io/intelaipg/intel-optimized-tensorflow
b234f539f7a1: Already exists
55172d420b43: Already exists
5ba5bbeb6b91: Already exists
43ae2841ad7a: Already exists
f6c9c6de4190: Already exists
5624105f79a2: Pull complete
1cbdffd12405: Pull complete
4111af644df8: Pull complete
060d55811dc3: Pull complete
41163fa89121: Pull complete
33ccfe03d160: Pull complete
eb5cd31a3268: Pull complete
b5fae668ebb8: Pull complete
5d9183ba9a31: Pull complete
d94ff583c6b6: Pull complete
Digest: sha256:fad4fdf1e45b38aacdb4b4f2767d65586b7e2dcf410c182e5676e70cf1602ed7
Status: Downloaded newer image for docker.io/intelaipg/intel-optimized-tensorflow:latest
[I 18:23:06.195 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[W 18:23:06.221 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[I 18:23:06.229 NotebookApp] Serving notebooks from local directory: /notebooks
[I 18:23:06.229 NotebookApp] 0 active kernels
[I 18:23:06.229 NotebookApp] The Jupyter Notebook is running at:
[I 18:23:06.229 NotebookApp] http://e4578d4d941e:8888/?token=d525b8e865d3453076e45f552d3f5943d7c170736dcbfc15
[I 18:23:06.229 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 18:23:06.230 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
```

The 'latest' and all other tags that don't have '`devel`' in them don't open an interactive terminal by default.

You can force open an interactive terminal by adding '`/bin/bash`' to the end of the `docker` run command. For example:

```
docker run -ti intelaipg/intel-optimized-tensorflow:latest
/bin/bash
```

```
[root@localhost TensorFlow_container]# docker run -it intelaipg/intel-optimized-tensorflow:latest /bin/bash
Unable to find image 'intelaipg/intel-optimized-tensorflow:latest' locally
Trying to pull repository docker.io/intelaipg/intel-optimized-tensorflow ...
latest: Pulling from docker.io/intelaipg/intel-optimized-tensorflow
8ee29e426c26: Pull complete
6e83b260b73b: Pull complete
e26b65fd1143: Pull complete
40dca07f8222: Pull complete
b420ae9e10b3: Pull complete
91bd4861698b: Pull complete
eb7fa39f11a1: Pull complete
377bcf4cd77f: Pull complete
9a9d21359ea2: Pull complete
2597e63e35d2: Pull complete
db5d788188eb: Pull complete
8aa4e84fe910: Pull complete
66cf7abb5492: Pull complete
a0ff3fca79cb: Pull complete
2c0806ae8fb7: Pull complete
Digest: sha256:a659d8acd7741056a3bd7835d92bd3877b35ea6a5bf2090ebb219c14330cd3cb
Status: Downloaded newer image for docker.io/intelaipg/intel-optimized-tensorflow:latest
root@3a3f0f324146:/notebooks#
```

Step 9. Get the Latest Benchmarks

Obtain the current TensorFlow benchmarks code from GitHub:

```
git clone https://github.com/tensorflow/benchmarks
cd benchmarks/scripts/tf_cnn_benchmarks
```
Run tf_cnn_benchmarks.py as explained below.

Note: The container is a light image; you will have to install basic Linux packages like yum, wget, vi, etc. on to the Docker container image. The authors ran the following steps before cloning the benchmark. The container image is based on Ubuntu.

```
apt-get update

apt-get install vim –y

apt-get install yum

apt-get install git
```

After making the necessary updates/changes to your container, exit and save the changes on the local version of the image by using:

```
docker commit <container_ID> <name_you_like>
```

Container_ID here is the ID provided when you initially ran the container.

Example:

```
[root@localhost TensorFlow_container]# docker commit 3a3f0f324146 tf_1.10_BZia_rev1
repository name must be lowercase
[root@localhost TensorFlow_container]# docker commit 3a3f0f324146 tf_1.10_bzia_rev1
sha256:60df6b3d468a8bc065d463c2d8048ac1c240d5f3d0bb7aa06121f57a5f210213
[root@localhost TensorFlow_container]# docker images
REPOSITORY                                    TAG                 IMAGE ID            CREATED             SIZE
tf_1.10_bzia_rev1                             latest              60df6b3d468a        9 seconds ago       2.07 GB
docker.io/intelaipg/intel-optimized-tensorflow   latest              4dc0f350e96d        3 weeks ago         1.91 GB
docker.io/hello-world                         latest              2cb0d9787c4d        7 weeks ago         1.85 kB
[root@localhost TensorFlow_container]#
```

Step 10: Running TensorFlow* Benchmark

This section covers run commands needed to run TensorFlow CNN benchmarks.

```
cd benchmarks/scripts/tf_cnn_benchmarks

python tf_cnn_benchmarks.py --forward_only=True --device=cpu --
mkl=True --kmp_blocktime=0 --nodistortions --batch_size=32 --
model=inception3 --data_format=NCHW  --num_intra_threads=4  --
num_inter_threads=1
```

Use the commands given in the TensorFlow* Performance Guide to learn how to get the best CPU optimized numbers. The example command above is for inceptionV3 model, but other models within the tf_cnn_benchmark directory can also be used.

Multiple Node Installation:

As of this writing, no publicly realized Intel-optimized Docker container was available for multiple nodes, so no step-by-step, tested instructions for that are included. However, published documents such as Horovod distributed training on Kubernetes using MLT are recommended reading for distributed training on Kubernetes. Additionally, to deploy TensorFlow via containers on multiple nodes, review the whitepaper Best known methods for scaling deep learning with TensorFlow* on Intel® Xeon® processor-based clusters as well as Nauta. This document provides steps to create singularity containers using a multimode environment, and then deploy/run those singularity containers.

Option 3: AI in Cloud

Single Node Installation:

Various cloud service providers (CSPs) can be used to deploy AI workloads via the cloud. This document cites a few of the major CSPs as examples for deploying AI in the cloud.

This example is for Amazon Web Services (AWS)* but you can use a CSP of your choice.

Source: AWS Deep Learning Tutorial

Step1: Sign in to AWS Management Console

Sign in to your AWS Management console with your username and password. Then select EC2 instance.

Step2: Configure your instance

1. Choose EC2 instance and click Launch Instance
2. Select an AWS deep learning AMI
   As mentioned in TensorFlow* Performance Guide, Amazon Machine Instance (AMI) is available for both Ubuntu and Amazon Linux. Choose the ideal fit for your application. This guide selected Deep Learning AMI (Ubuntu).

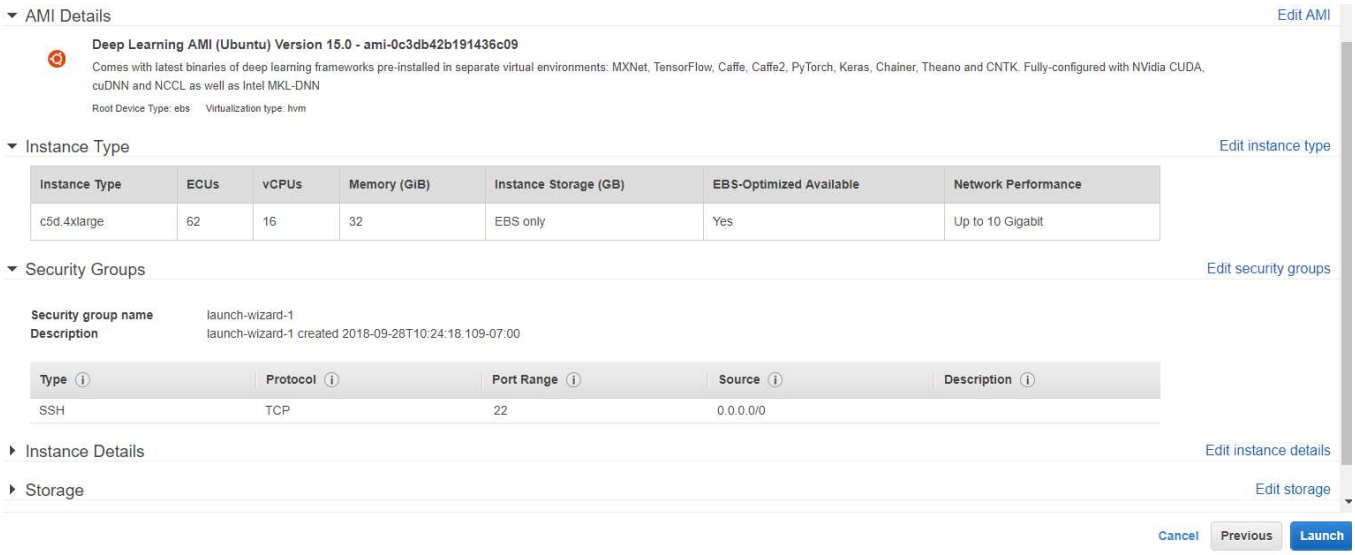
3. Choose the instance type for deep learning and deployment needs. Select Compute Optimized Instance (c5), to get CPU optimized hardware and software. Then click Review and Launch.

Example:

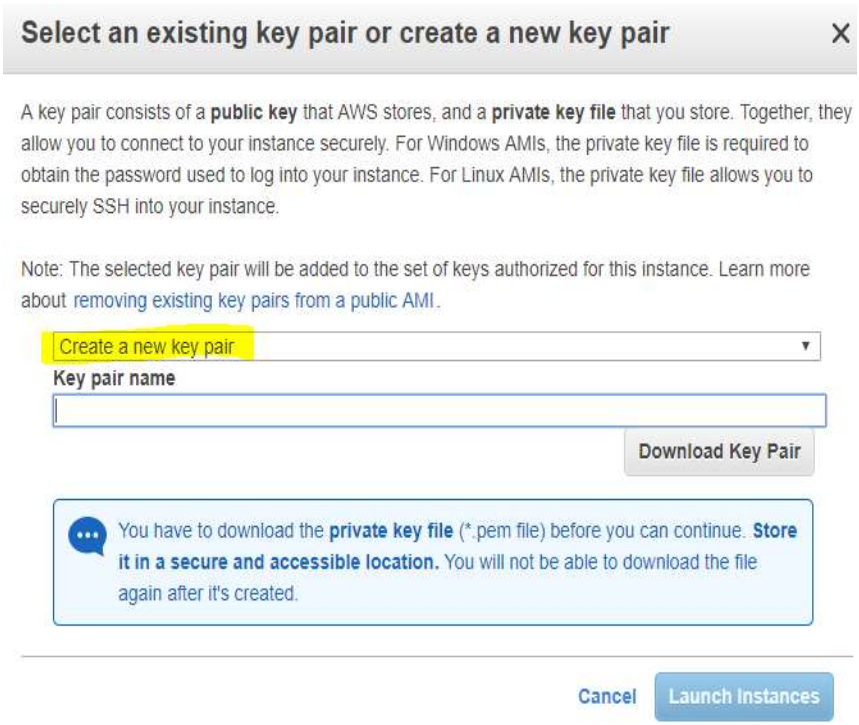
☑	Compute optimized	c5d.4xlarge	16	32	1 x 400 (SSD)	Yes	Up to 10 Gigabit	Yes
☐	Compute optimized	c5d.9xlarge	36	72	1 x 900 (SSD)	Yes	10 Gigabit	Yes
☐	Compute optimized	c5d.18xlarge	72	144	2 x 900 (SSD)	Yes	25 Gigabit	Yes
☐	Compute optimized	c5.large	2	4	EBS only	Yes	Up to 10 Gigabit	Yes
☐	Compute optimized	c5.xlarge	4	8	EBS only	Yes	Up to 10 Gigabit	Yes
☐	Compute optimized	c5.2xlarge	8	16	EBS only	Yes	Up to 10 Gigabit	Yes
☐	Compute optimized	c5.4xlarge	16	32	EBS only	Yes	Up to 10 Gigabit	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

4. Choose Launch on the review page



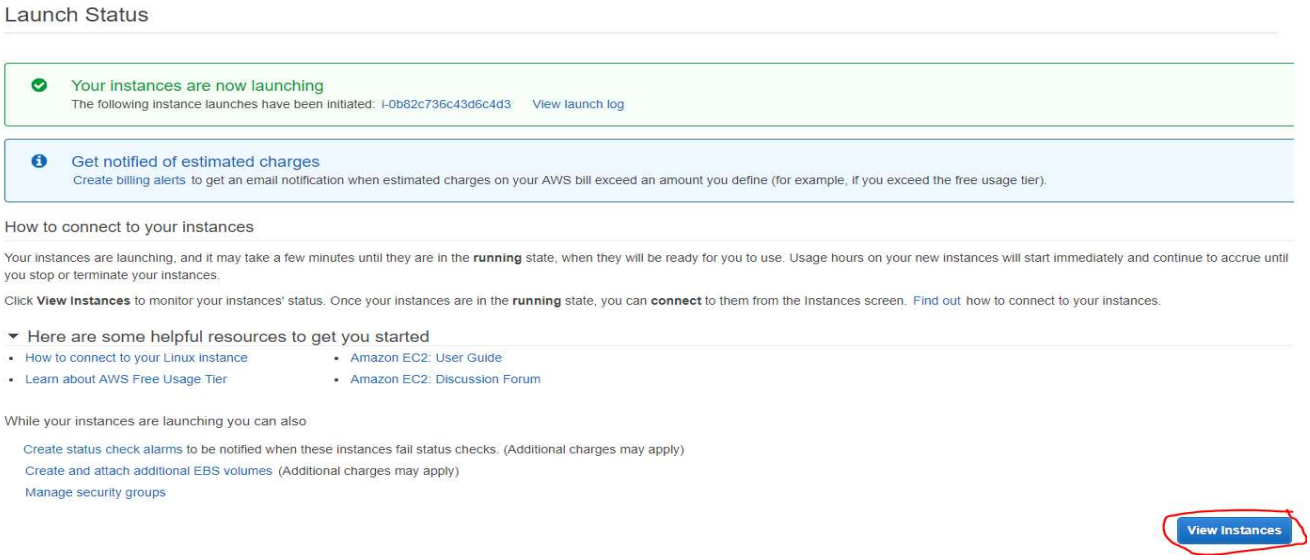
5. Create a private key file by selecting Create New Key Pair, and download it to a safe location. Then launch the instance. You will see screen shot as follows:



Note: You may get a message saying "your account is currently being verified." Verification usually takes less than two hours. You can retry to launch the instance after 30 minutes.

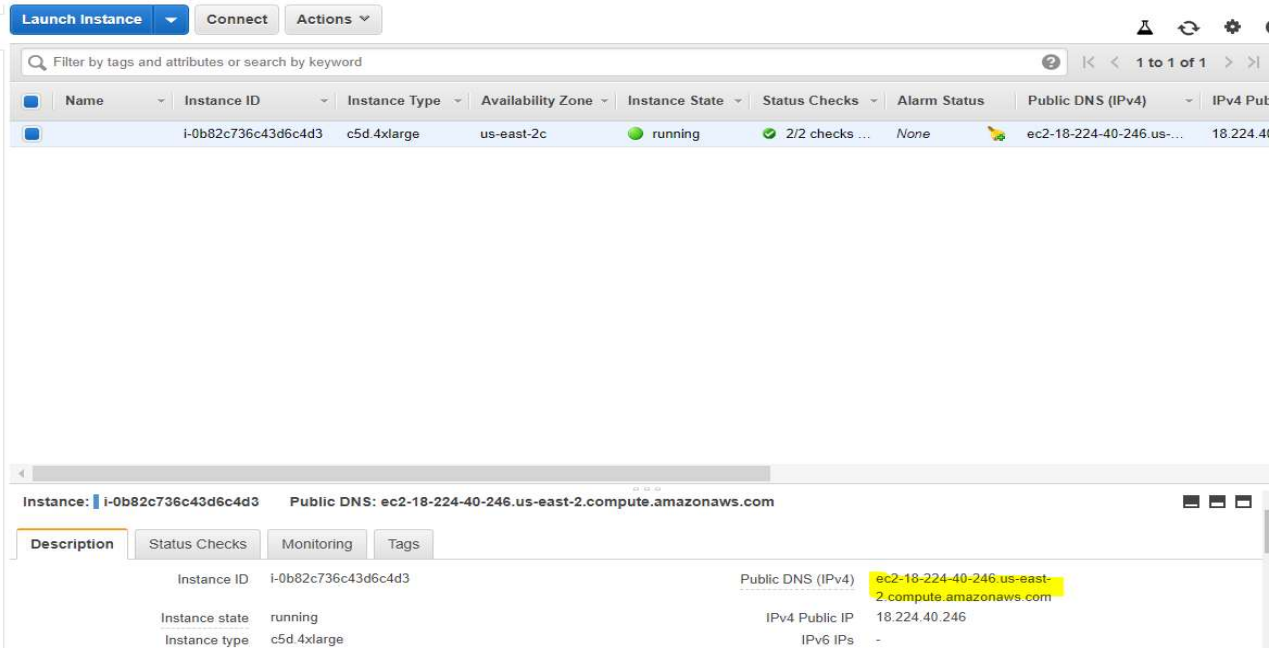
6. Click view instance to see your instance.

Example:



7. After clicking View Instance, find and copy your instance's public DNS.

Example screen shot:



Step 3: Connect to your instance

To start using the command line terminal to communicate with the instance on AWS using Windows*, use a command prompt or download Git for Windows.[25]

a) Following the steps described in [24], open the command terminal.
b) Change to the directory where your security key is located.
c) Change the permissions on the key pair file.
d) Connect to your instance using SSH.

Example:

```
cd /user/xzy/Downloads/

chmod 0400 <key_file_name.per>

ssh -l localhost:8888:localhost:8888 -i <key_file_name.per>
ubuntu@<your_instance_DNS_that_you_copied>
```

Note: If you are on a corporate network, it may be necessary to use a proxy to connect to your instance.

Using Putty:

1. Download and install PuTTY from the PuTTY download page
2. PuTTY natively doesn't support .pem file generated by the AWS EC2, so convert the .pem key file to the required PuTTY format (.ppk).
3. Convert your private key:
    a. Launch Puttygen
    b. Under type of key to generate, choose RSA. When using older versions of PuTTYgen, choose SSH-2 RSA

c. Choose Load. By default, PuTTY will list only .ppk files. Select All Files from the dropdown menu to view .pem file

d. Select the .pem file for the AWS key pair you specified when you launched your instance, then choose Open. Choose OK to dismiss the dialog box.

e. Select Save Private Key to save the key in a format PuTTY can use. PuTTYgen displays a warning about saving the key without a passphrase. Choose Yes.

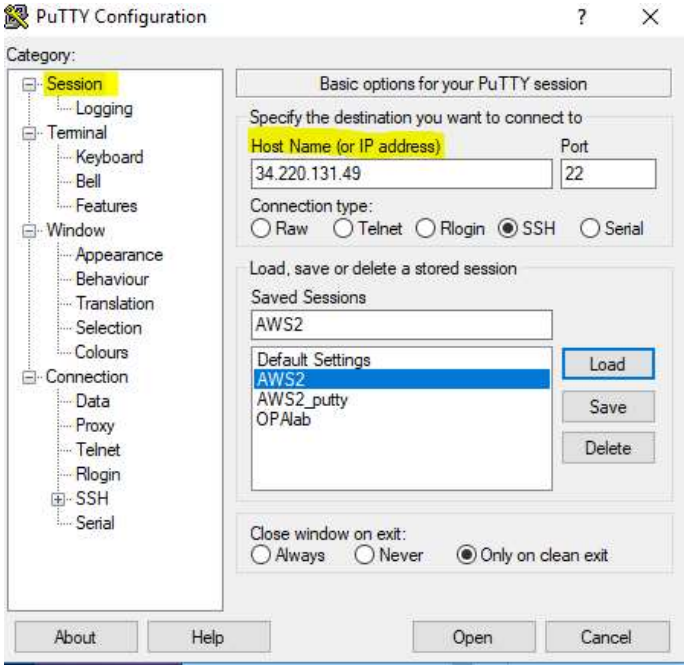
f. Specify the same name for the key you used for the key pair. PuTTY automatically adds the .ppk file extension.

4. Start your PuTTY session:

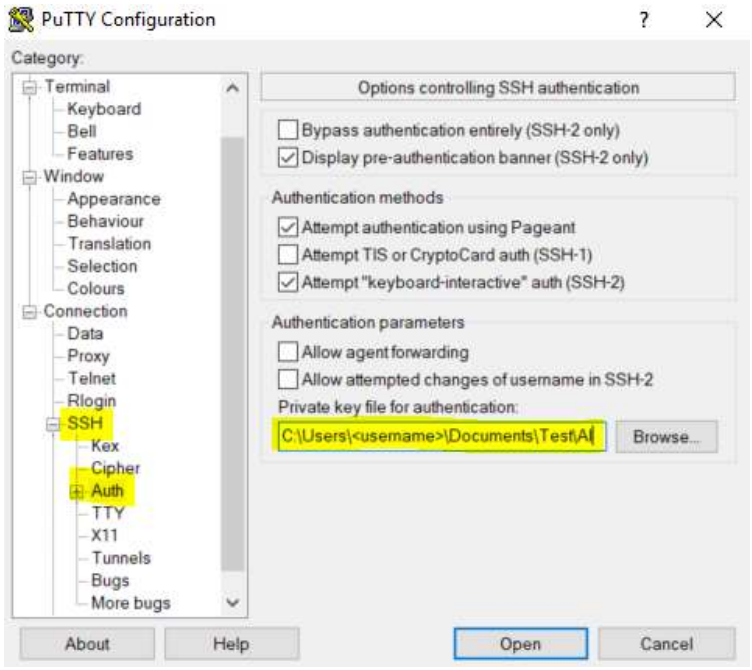
a. Launch PuTTY.

b. Click on Session and in hostname add the IPv4 Public IP address of your EC2 instance.

Example:

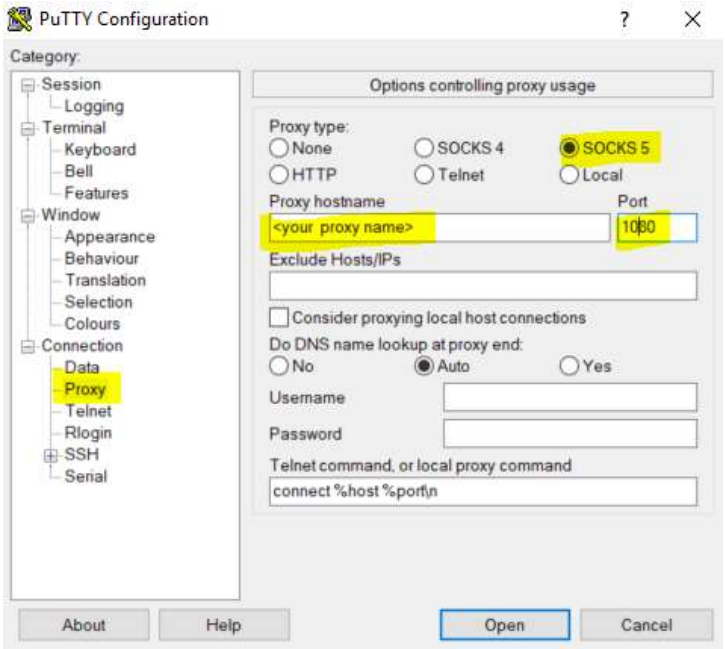


c. Now click on SSH under connection, expand it, and click on Auth. In Auth, add the link to your .ppk file created from the .pem file. Example:

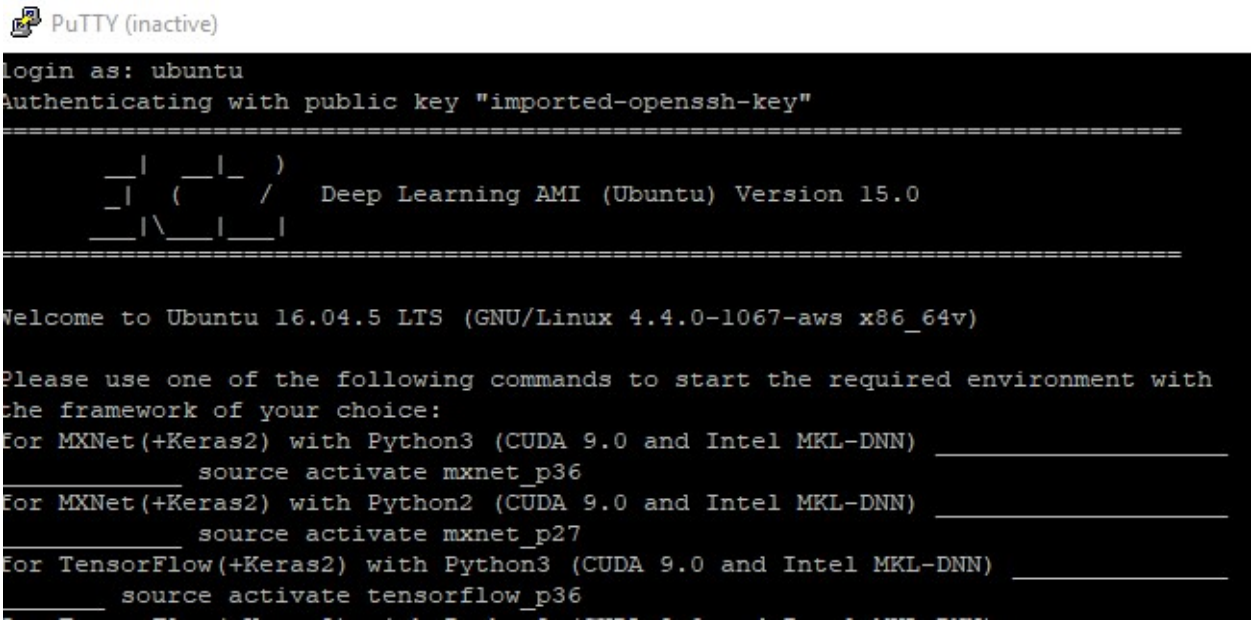


d. If you are on a corporate network, click on Proxy, Select Socks5, enter the name of your proxy network, add port 1080

Example:



e. Save the session under a name, if you like, under the Session tab.
f. Once saved, click Open.
g. For Login type Ubuntu.
h. The session should start successfully like example snapshot below:



Step 4: Run your deep learning framework

Run deep learning workload using Jupyter* or directly on the command terminal.

For Jupyter: type jupyter notebook

Copy the URL indicated to access your notebook and start using a deep learning framework.

For terminal:

This whitepaper uses containers to install the Intel Optimization for TensorFlow framework and run benchmark examples. The AWS instance running should already have Docker installed.

a. Type the following command to check if Docker is installed.

```
apt-cache policy docker-ce
```

You should see Docker Installed. If it says Installed: (none), install Docker on your instance. To install Docker on Ubuntu, follow the steps here or on a similar document.

```
ubuntu@ip-172-31-47-94:~/beenish$ apt-cache policy docker-ce
docker-ce:
  Installed: 18.06.1~ce~3-0~ubuntu
  Candidate: 18.06.1~ce~3-0~ubuntu
  Version table:
 *** 18.06.1~ce~3-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
        100 /var/lib/dpkg/status
     18.06.0~ce~3-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
     18.03.1~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
     18.03.0~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
```

b. To see Docker running, type:

```
sudo systemctl status docker
```

```
ubuntu@ip-172-31-47-94:~/beenish$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2018-10-03 17:48:52 UTC; 5 days ago
     Docs: https://docs.docker.com
 Main PID: 1283 (dockerd)
    Tasks: 79
   Memory: 165.3M
      CPU: 14min 22.162s
   CGroup: /system.slice/docker.service
           ├─1283 /usr/bin/dockerd -H fd://
           └─1524 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

Run Docker hello-world to check if Docker is installed correctly.

c. Pull the Intel Optimization for TensorFlow Docker container using the command below. Follow steps under option 2 to view details.

```
docker pull docker.io/intelaipg/intel-optimized-
tensorflow:latest-devel-mkl
```

To see a list of all available Docker images on your system, type following command:

```
docker images
```

```
ubuntu@ip-172-31-47-94:~/beenish$ docker pull docker.io/intelaipg/intel-optimized-tensorflow:latest-devel-mkl
latest-devel-mkl: Pulling from intelaipg/intel-optimized-tensorflow
8ee29e426c26: Pull complete
6e83b260b73b: Pull complete
e26b65fd1143: Pull complete
40dca07f8222: Pull complete
b420ae9e10b3: Pull complete
c64f3de286db: Pull complete
507efe46106f: Pull complete
1269cfeef87f: Pull complete
73d3a3b039fc: Pull complete
b3aec0e4d23b: Pull complete
8c33477c03f7: Pull complete
1766942a54eb: Pull complete
81ce72be8384: Pull complete
5c03eacfa02f: Pull complete
709907265e4b: Pull complete
d7422f948262: Pull complete
cd6e10816369: Pull complete
3b2e3516c2a7: Pull complete
3ad59c05c44b: Pull complete
174af12f4061: Pull complete
Digest: sha256:0c93f6f34e14951b9128453362la8fc47cde87240bc67abff45dlb6df3b48a57
Status: Downloaded newer image for intelaipg/intel-optimized-tensorflow:latest-devel-mkl
ubuntu@ip-172-31-47-94:~/beenish$ docker images
REPOSITORY                              TAG                 IMAGE ID            CREATED             SIZE
hello-world                             latest              4ab4c602aa5e        4 weeks ago         1.84kB
intelaipg/intel-optimized-tensorflow    latest-devel-mkl    40b5d30f8b17        2 months ago        3.03GB
```

d. Running Intel Optimization for TensorFlow Docker image.

The 'latest' and all tags that don't have 'devel' in them don't open an interactive terminal by default.

For Jupyter notebook, run following command:

```
docker run -it -p 8888:8888 intelaipg/intel-optimized-tensorflow
```

Go to your browser on http://localhost:8888/

For command terminal, run following command:

```
docker run -ti intelaipg/intel-optimized-tensorflow:latest
/bin/bash
```

The latest tag will automatically load the latest version of the optimized container.

```
ubuntu@ip-172-31-47-94:~/beenish$ docker run -ti intelaipg/intel-optimized-tensorflow:latest /bin/bash
Unable to find image 'intelaipg/intel-optimized-tensorflow:latest' locally
latest: Pulling from intelaipg/intel-optimized-tensorflow
8ee29e426c26: Already exists
6e83b260b73b: Already exists
e26b65fd1143: Already exists
40dca07f8222: Already exists
b420ae9e10b3: Already exists
91bd4861698b: Pull complete
eb7fa39f11a1: Pull complete
377bcf4cd77f: Pull complete
9a9d21359ea2: Pull complete
2597e63e35d2: Pull complete
db5d788188eb: Pull complete
8aa4e84fe910: Pull complete
66cf7abb5492: Pull complete
a0ff3fca79cb: Pull complete
2c0806ae8fb7: Pull complete
Digest: sha256:a659d8acd7741056a3bd7835d92bd3877b35ea6a5bf2090ebb219c14330cd3cb
Status: Downloaded newer image for intelaipg/intel-optimized-tensorflow:latest
root@4f0efcf08f77:/notebooks#
```

Step 5: Running TensorFlow benchmark

The TensorFlow container you pulled is a light image; install basic Linux packages such as yum, wget, vi etc. onto the Docker container image before running benchmarks. The author ran the following steps before cloning the benchmark. The container image is based on Ubuntu.

```
apt-get update

apt-get install vim –y

apt-get install yum

apt-get install git
```

Once the necessary updates/changes to the container are made, exit and save the changes on the local version of the image by using:

```
root@<container_ID>:/notebooks# exit

root@<container_ID>:/notebooks# docker commit <container_ID>
<name_you_like>
```

container_ID here is the ID given to you when you initially ran the container.

Example:

```
root@4f0efcf08f77:/notebooks# exit
exit
ubuntu@ip-172-31-47-94:~/beenish$ docker commit 4f0efcf08f77 tf_container_bzia_rev1
sha256:8ce021fcaf1db8b50911719c6510ac3589cca8fbd0aaa2ad9ad8479821acd0b0
ubuntu@ip-172-31-47-94:~/beenish$
```

Now run the new Docker image:

```
docker run –ti
<new_name_you_gave_to_container_saved_above>:latest /bin/bash
```

Example:

```
(python2) ubuntu@ip-172-31-47-94:~/beenish$ docker run -ti tf_container_bzia_rev1:latest /bin/bash
root@a8e0e796f9b8:/notebooks#
```

Obtain the current TensorFlow benchmarks code from GitHub:

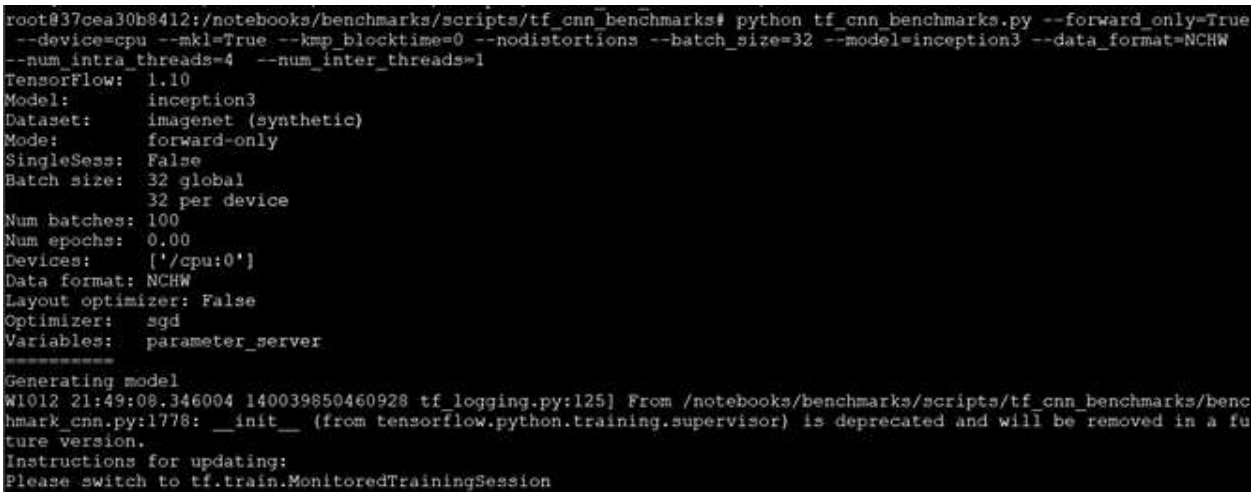
```
git clone
cd benchmarks/scripts/tf_cnn_benchmarks
```

This section discusses commands needed to run TensorFlow CNN benchmarks:

```
python tf_cnn_benchmarks.py --forward_only=True --device=cpu --
mkl=True --kmp_blocktime=0 --nodistortions --batch_size=32 --
model=inception3 --data_format=NCHW  --num_intra_threads=4  --
num_inter_threads=1
```

The commands given here provide exact details for achieving the best CPU optimized numbers. The example command above is for inceptionV3 model, but other models within the `tf_cnn_benchmark` directory can also be used.

Here's an example screenshot after running the command above:



Note: When running the Python command and this error message appears: "No module experimental.ops," it probably means the version of container TensorFlow is not compatible with the version used for benchmarks.

Example: if benchmarks are compatible with TensorFlow v1.12 and your container is for TensorFlow v1.10, download the compatible benchmark using the following command:

```
git clone -b cnn_tf_v1.10_compatible
https://github.com/tensorflow/benchmarks
```

Step 6: Terminate your instance

Multiple Node Installation:

This section describes steps for running distributed TensorFlow using Google Cloud Platform* with Google Machine Learning (ML) Engine. It details how to get an instance running on Google ML Engine,

deploy TensorFlow, and run an example training on an all-CPU, multi-node setting, which is assigned by selecting standard_1 tier when running the distributed training. The majority of steps described in this section are from the Google ML Engine getting started guide.

Using Google Cloud Platform* (GCP)

Source: TensorFlow Getting Started

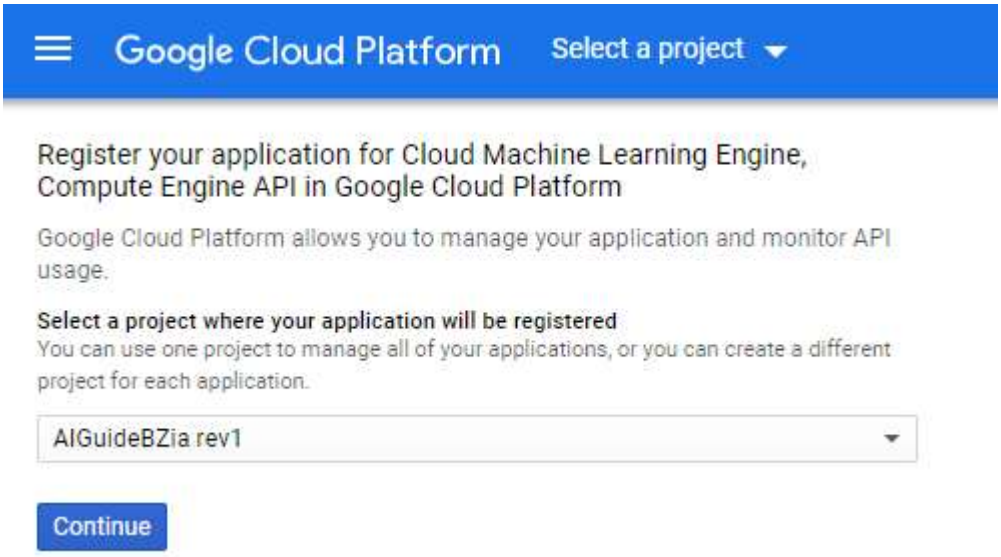
Step 1: Sign in to your Google account.

Step 2: Select or create a GCP project via "go to the manage resources page."

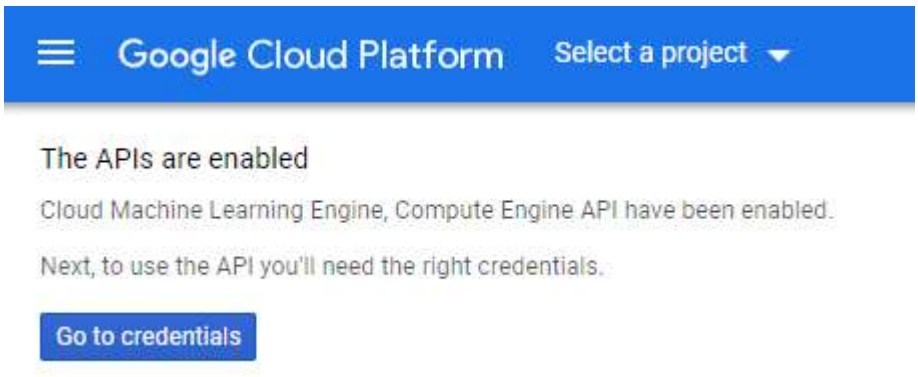
Step 3: Enable billing for your project by following these instructions.

Step 4: Enable the Cloud Machine Learning Engine and Compute Engine APIs by clicking this link:

Expect to see a screen like below. Select your project and click "Continue." Enabling APIs takes a few minutes.



Once enabling APIs is complete, expect to see a screen like the following:



Step 5: Set up authentication

a) In the GCP console, go to the Create service account key page.

b) From the **service account** dropdown list, select **New Service Account**.

c) In the **Service Account Name** field, enter a name.

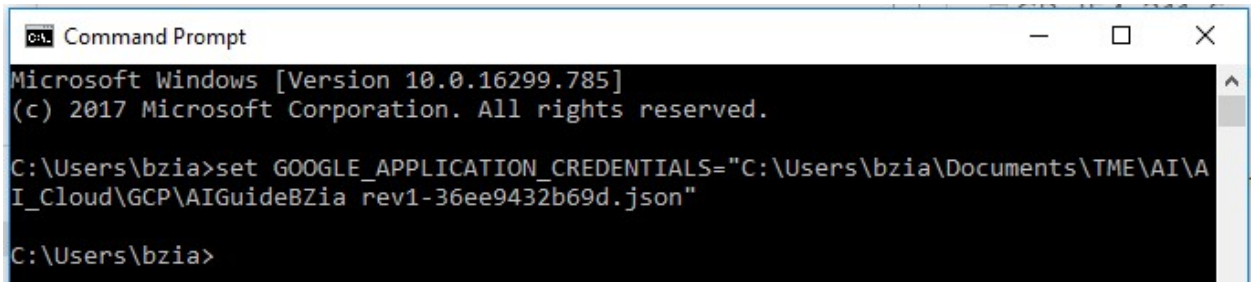
d) From the **Role** drop-down list, select **Project** > **Owner**.

e) Click **Create**. A JSON file that contains your key downloads to your computer.

Step 6: Set the environment variable ***GOOGLE_APPLICATION_CREDENTIALS*** to the file path of the JSON file that contains your service account key. This variable only applies to your current shell session; if you open a new session, set the variable again.

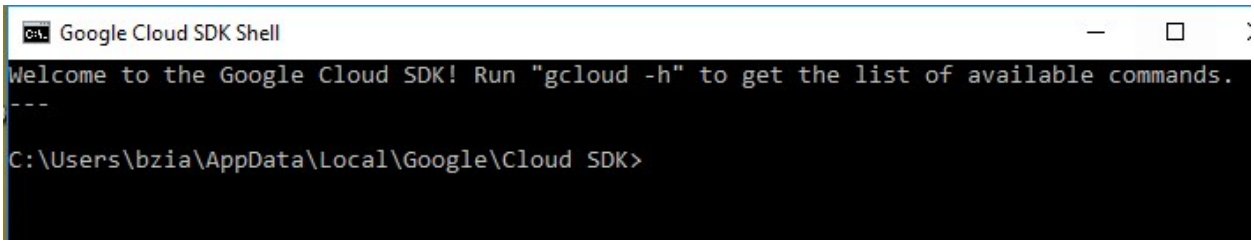
For Windows, using command prompt will be something like the following:

```
set GOOGLE_APPLICATION_CREDENTIALS= "<PATH>"
```

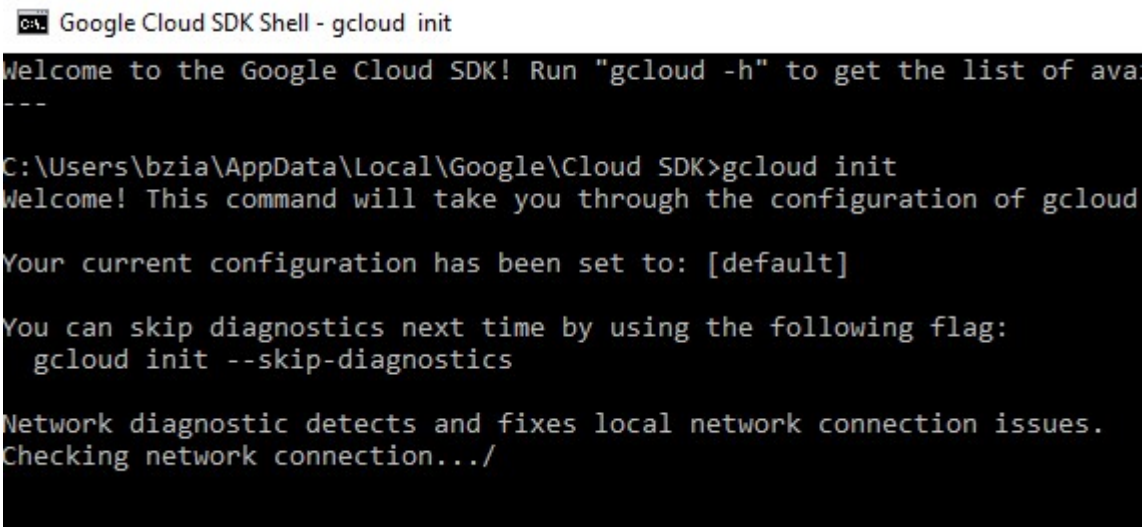


Step 7: Install and initialize the cloud SDK per instructions here.

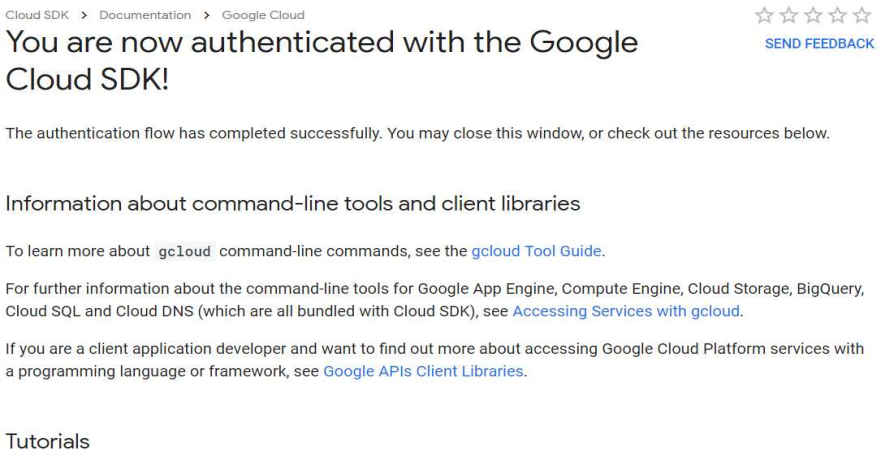
a. Download the Cloud SDK Installer, making sure to sign in with correct Google credentials.
   Note: If you get a Download Failed: connecting to host message and you are possibly behind a corporate firewall, follow the steps here to install from an archived version if you are unable to get around the firewall.
b. Launch the installer and follow the instructions on prompt.
c. After installation is completed, accept the following options:
   - Start cloud SDK shell. (For Windows users, if it hasn't started by default, find and click on the installed SDK.) It should open a command prompt as shown in the following:



   - Run gcloud init (For Windows, type gcloud init in the terminal window as follows)

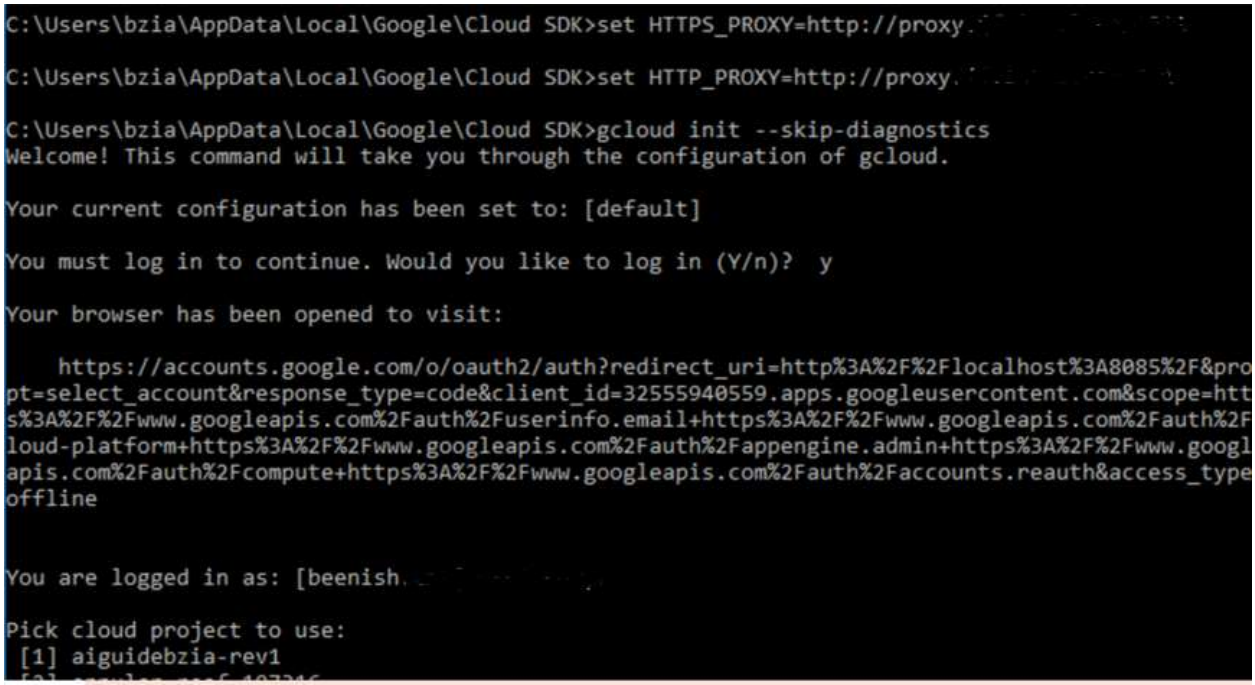


Once authentication is complete, expect to see a web browser page open and display a screen like the following:

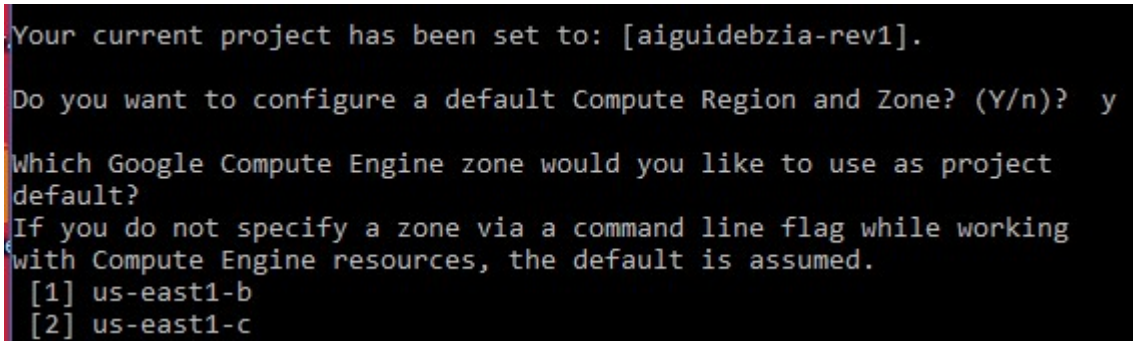


Note: If you have a network proxy, set it up as part of the glcoud setup using the correct HTTPS_PROXY and HTTP_PROXY address

Example:



d. After setting a cloud project, the system will ask to set a specific time zone. Selecting this time zone is critical because later during distributed training on the cloud, you will have to create a region for your cloud storage bucket, which should match the time zone you select for running your ML Cloud Engine.



e. Once complete, expect to see a message that your Google Cloud SDK is configured and ready to use.
Example:

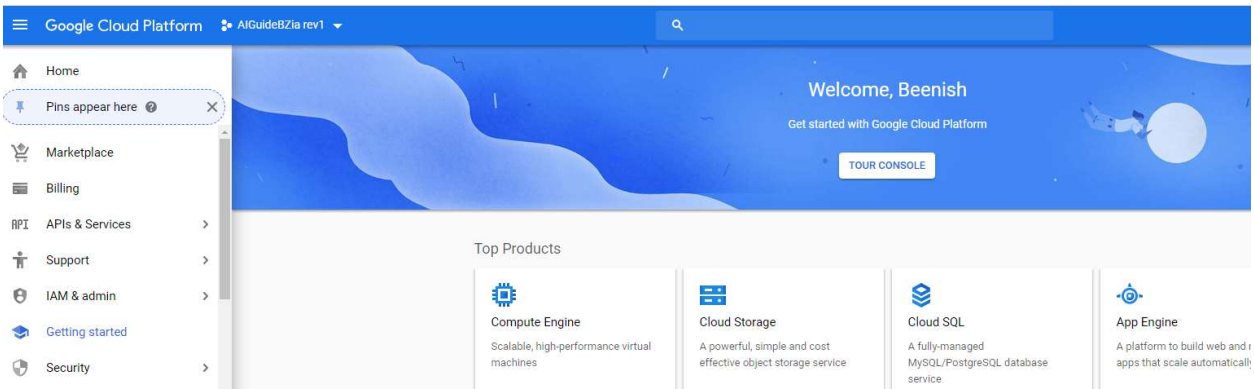
```
Your project default Compute Engine zone has been set to [us-west1-b].
You can change it by running [gcloud config set compute/zone NAME].

Your project default Compute Engine region has been set to [us-west1].
You can change it by running [gcloud config set compute/region NAME].

Created a default .boto configuration file at [C:\Users\bzia\.boto]. See this file and
[https://cloud.google.com/storage/docs/gsutil/commands/config] for more
information about configuring Google Cloud Storage.
Your Google Cloud SDK is configured and ready to use!

* Commands that require authentication will use beenish.zia@intel.com by default
* Commands will reference project `aiguidebzia-rev1` by default
* Compute Engine commands will use region `us-west1` by default
* Compute Engine commands will use zone `us-west1-b` by default

Run `gcloud help config` to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configurations if you w
ork with multiple accounts and/or projects.
Run `gcloud topic configurations` to learn more.

Some things to try next:

* Run `gcloud --help` to see the Cloud Platform services you can interact with. And run `gcloud
help COMMAND` to get help on any gcloud command.
* Run `gcloud topic -h` to learn about advanced features of the SDK like arg files and output fo
rmatting

C:\Users\bzia\AppData\Local\Google\Cloud SDK>
```

Step 8: Setting up the environment

Instructions in this section are taken from the Google Cloud AI and Machine Learning Products page. Google has both MAC OS and Cloud Shell (for macOS*, Linux, and Windows) instructions to set up your environment locally. However, this section covers only the steps from Cloud Shell as done on a Windows machine.

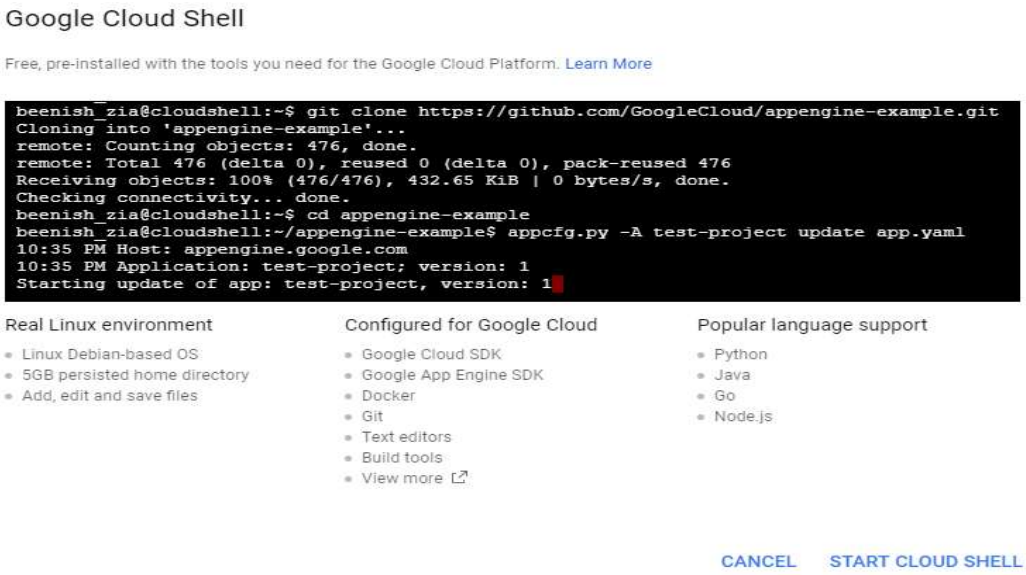
1. Open the Google Cloud Platform Console

A web browser page like the following screenshot should open. You can select your respective project from the dropdown menu on the top left.

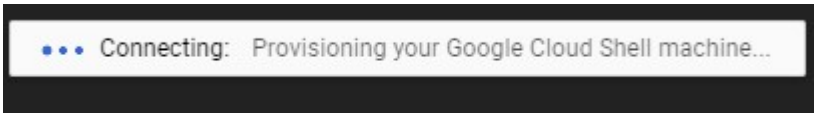


2. Click the Activate Google Cloud Shell button at the top of the console window.

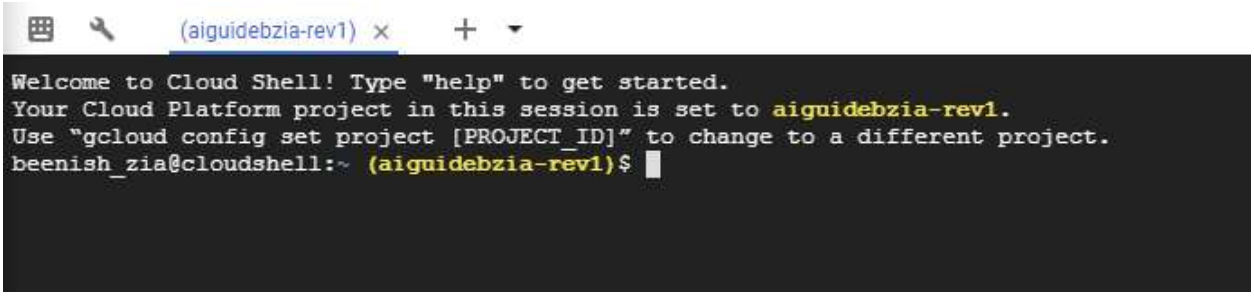
A window similar to the following screenshot opens. Click on Start Cloud Shell to begin.



Wait for the cloud shell machine to start.



A cloud shell session should open inside a new frame at the bottom of the Google Cloud platform page. It will look like the following:



3. If you did not select your project ID in Google Cloud Platform page already, then you can set it up now or change to a different project ID using the command:

   gcloud config set project <project-ID>

Step 9: Verify the google cloud SDK components

1. List your models:

   ```
   gcloud ml-engine models list
   ```

If you have not created any models before, the command returns an empty list.

```
beenish_zia@cloudshell:~ (aiguidebzia-rev1)$ gcloud ml-engine models list
Listed 0 items.
beenish_zia@cloudshell:~ (aiguidebzia-rev1)$
```

2. If you have installed gcloud previously, update gcloud:

   ```
   gcloud components update
   ```

Install TensorFlow*

a. To install TensorFlow, run the following command:

   ```
   pip install –user –upgrade tensorflow
   ```

```
beenish_zia@cloudshell:~ (aiguidebzia-rev1)$ pip install --user --upgrade tensorflow
Requirement already up-to-date: tensorflow in /usr/local/lib/python2.7/dist-packages (1.12.0)
```

By default, TensorFlow will probably be installed so you will see a message about requirement already up to date, with the version of TensorFlow.

Run a simple TensorFlow Python program (*Optional*)

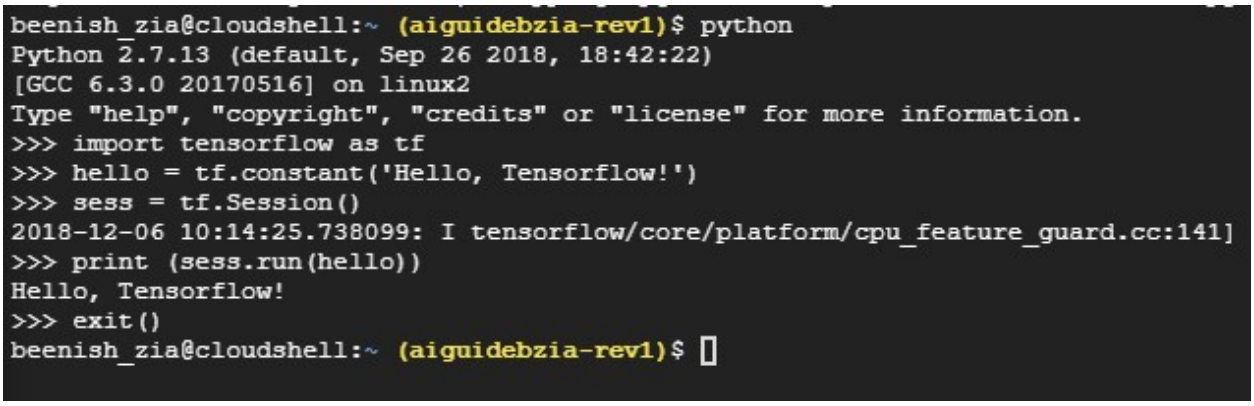
Try to run the following basic Python program to gain confidence in writing and testing your installation of TensorFlow.

```
import tensorflow as tf
hello= tf.constant('Hello, Tensorflow!')
sess = tf.Session()
print(sess.run(hello))
```

If successful, the system outputs:

```
Hello, Tensorflow!
>>> exit()
```

Your test should resemble the following screenshot:



As stated on the Google Cloud AI and ML Products page, from where these instructions are taken, the Cloud ML Engine runs Python 2.7 by default and the sample code for this section uses Python* 2.7. You can check Google Cloud AI and ML Products page to further learn how to use Python 3.5 for submitting jobs.

Step 10: Download the code for the example

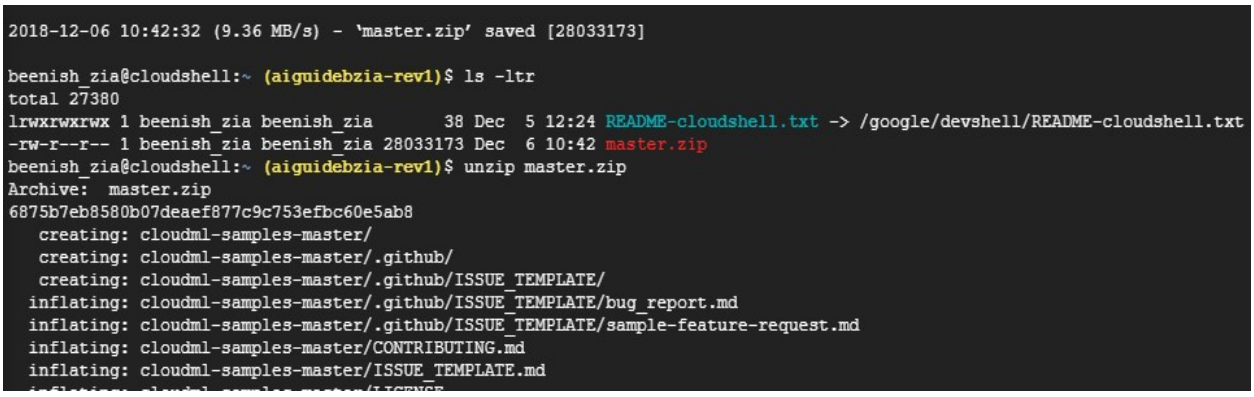
This document shows steps for Cloud Shell running on Windows OS. Good AI and ML Learning Products: Getting Started Google Cloud AI and ML Products page, provides steps for MacOS as well as Cloud Shell.

1. Download the sample zip file from the GitHub Repository:

Unzip the sample zip file to extract the cloudml-samples-master directory:

```
unzip master.zip
```

Example screenshot:



2. Navigate to the cloudml-samples-master → census → estimator directory. The commands in the section of the guide must be run from the **estimator** directory.

```
cd cloudml-samples-master/census/estimator
```

Step 11: Get training data

Google hosts a public Cloud Storage bucket, where the relevant data files, **adult.data**, and **adult.test** exist for this section.

1. Create a data directory and download the data to the estimator directory:

   ```
   mkdir data
   ```

   ```
   gsutil –m cp gs://cloud-samples-data/ml-engine/census/data/* data/
   ```

2. Set the TRAIN_DATA and EVAL_DATA variable to the file paths.
   Example**:**

   ```
   TRAIN_DATA=<local path>/data/adult.data.csv
   EVAL_DATA=<local path>/data/adult.test.csv
   ```

Step 12: Install dependencies

TensorFlow is installed on Cloud Shell, but the sample in this section is based on TensorFlow 1.10. Hence, you must run the sample's requirements.txt file to ensure you are using the same version of TensorFlow and other dependencies as required by the sample in this section.

You should be in the `clouldml-samples-master/census/estimator` directory.

You will find `requirements.txt` file one directory above.

```
beenish_zia@cloudshell:~/cloudml-samples-master/census/estimator (aiguidebzia-rev1)$ ls -ltr ..
total 56
drwxr-xr-x 4 beenish_zia beenish_zia  4096 Dec  6 09:25 tftransformestimator
-rw-r--r-- 1 beenish_zia beenish_zia   314 Dec  6 09:25 test.json
-rw-r--r-- 1 beenish_zia beenish_zia   241 Dec  6 09:25 test.csv
drwxr-xr-x 3 beenish_zia beenish_zia  4096 Dec  6 09:25 tensorflowcore
-rwxr-xr-x 1 beenish_zia beenish_zia  1788 Dec  6 09:25 sample.sh
-rw-r--r-- 1 beenish_zia beenish_zia   199 Dec  6 09:25 requirements.txt
-rw-r--r-- 1 beenish_zia beenish_zia 14934 Dec  6 09:25 README.md
drwxr-xr-x 3 beenish_zia beenish_zia  4096 Dec  6 09:25 keras
-rw-r--r-- 1 beenish_zia beenish_zia   564 Dec  6 09:25 hptuning_config.yaml
drwxr-xr-x 3 beenish_zia beenish_zia  4096 Dec  6 09:25 customestimator
drwxr-xr-x 4 beenish_zia beenish_zia  4096 Dec  6 13:24 estimator
beenish_zia@cloudshell:~/cloudml-samples-master/census/estimator (aiguidebzia-rev1)$
```

Now run:

```
Pip install --user –r ../requirements.txt
```

Step 13: Set up your cloud storage bucket

1. Specify a name for your new bucket. The name must be unique across all buckets in Cloud Storage.

   ```
   BUCKET_NAME =”<your bucket name>”
   ```

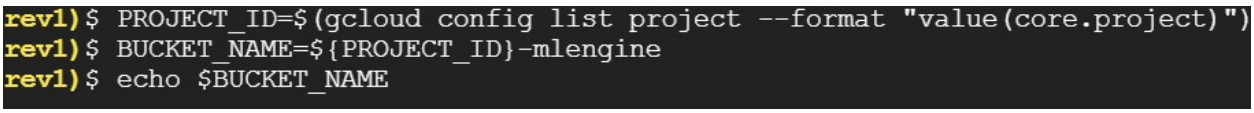
Note: To be sure your bucket name is unique, it's recommended to use your project name with `–mlengine`, as shown in the command below.

```
PROJECT_ID=$(gcloud config list project –format
“value(core.project)”)
BUCKET_NAME=${PROJECT_ID}-mlengine
```

2. Check the name for the bucket you created:

```
echo $BUCKET_NAME
```

Example screenshot:



3. Select a region for your bucket and set the environment variable:

```
REGION=<name of region>
```

Example:

```
REGION=us-west1
```

Note 1: Specify a unique region for your bucket; it cannot have a multi-region location. Find an available region at Google Cloud AI and ML Products page[29] for Cloud ML Engine.

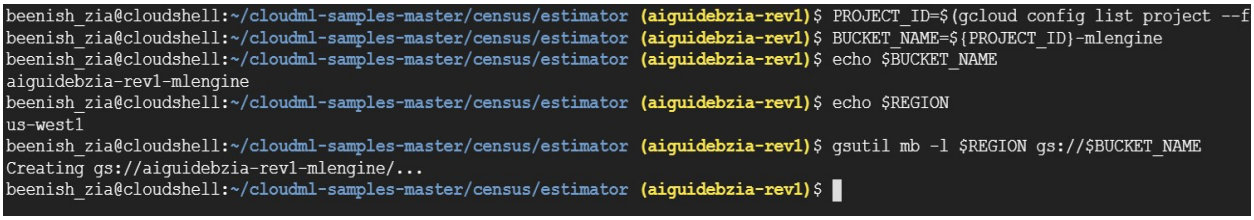
Note 2: Use the same region where you plan on running Cloud ML Engine jobs – the region you chose in step 7 of “using Google Cloud platform” sub-section.

Note 3: If you restart your session, you may have lost your environment settings for BUCKET_NAME and REGION. It is recommended to check the variable setting before going to the next step, especially doing a restart of the cloud shell.

4. Create the new bucket:

```
gsutil mb –l $REGION gs://$BUCKET_NAME
```

Example screenshot:



Step 14: Upload the data files to your cloud storage bucket

1. Use `gsutil` to copy the two files to your newly created cloud storage bucket:

```
gsutil cp -r data gs://$BUCKET_NAME/data
```

2. Point the TRAIN_DATA and EVAL_DATA variables to the file location in your cloud storage bucket:

```
TRAIN_DATA=gs://$BUCKET_NAME/data/adult.data.csv
EVAL_DATA=gs://$BUCKET_NAME/data/adult.test.csv
```

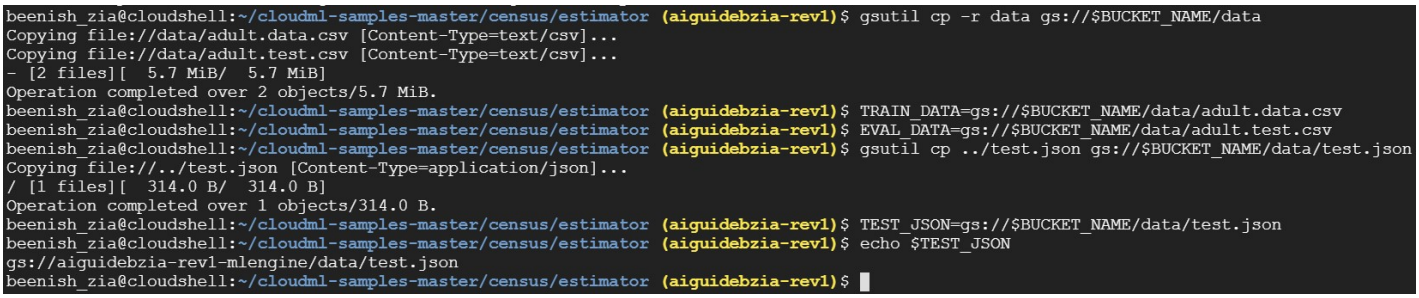
3. Copy the JSON test file to your cloud storage bucket using gsutil:

```
gsutil cp ../test.json gs://$BUCKET_NAME/data/test.json
```

4. Set the TEST_JSON to point to the file:

```
TEST_JSON=gs://$BUCKET_NAME/data/test.json
```

Here's an example screen capture of uploading data files to your cloud storage bucket:



Step 15: Run distributed training in the cloud

To run your training job in distributed mode in Google cloud, the commands are very similar to those used to run training on a single instance on Google cloud. This document didn't cover single instance training on Google cloud, but it can be found on the Google Cloud AI and ML Products page. The major difference for training in single instance vs. multiple is setting – scale-tier to correct tier compared to basic for a single instance. Find information on available scale tiers for Google Cloud ML Engine here.

1. Set a name for your distributed training job:

```
JOB_NAME=<your distributed training name>
```

Example:

```
JOB_NAME=census_dist_rev1
```

2. Create an `OUTPUT_PATH`. We recommend adding `JOB_NAME` to avoid reusing checkpoints between jobs.

Note You might have to redefine `BUCKET_NAME` if you've started a new shell session. Run echo `$BUCKET_NAME` to make sure variables are set correctly.

`OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_NAME`

Example:

```
$ JOB_NAME=census_dist_rev1
$ echo $BUCKET_NAME

$ OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_NAME
$
```

3. Run the following command to submit a distributed training job in the Google Cloud that uses multiple workers. This example uses **standard_1** scale tier to **use an all-CPU-based configuration.** The job can take a few minutes to start.

Place the `--scale-tier` above the -- that separates the user arguments from the command line arguments.

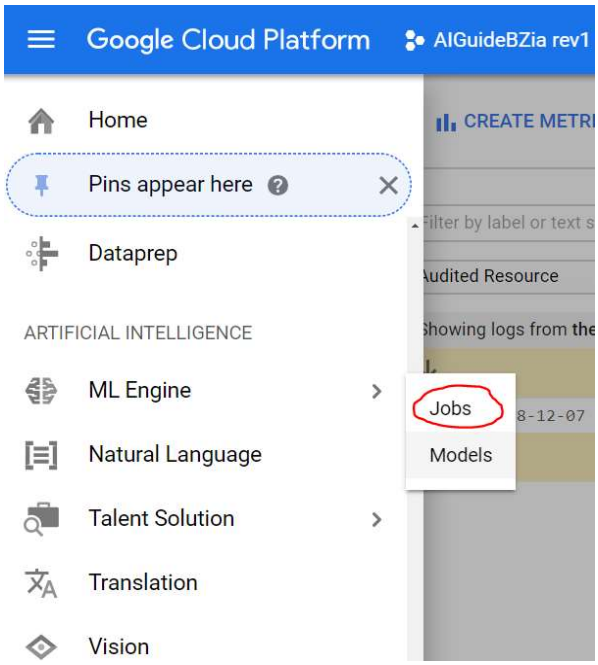
Example[31]

```
gcloud ml-engine jobs submit training $JOB_NAME \
    --job-dir $OUTPUT_PATH \
    --runtime-version 1.10 \
    --module-name trainer.task \
    --package-path trainer/ \
    --region $REGION \
    --scale-tier STANDARD_1 \
    -- \
    --train-files $TRAIN_DATA \
    --eval-files $EVAL_DATA \
    --train-steps 1000 \
    --verbosity DEBUG  \
    --eval-steps 100
```

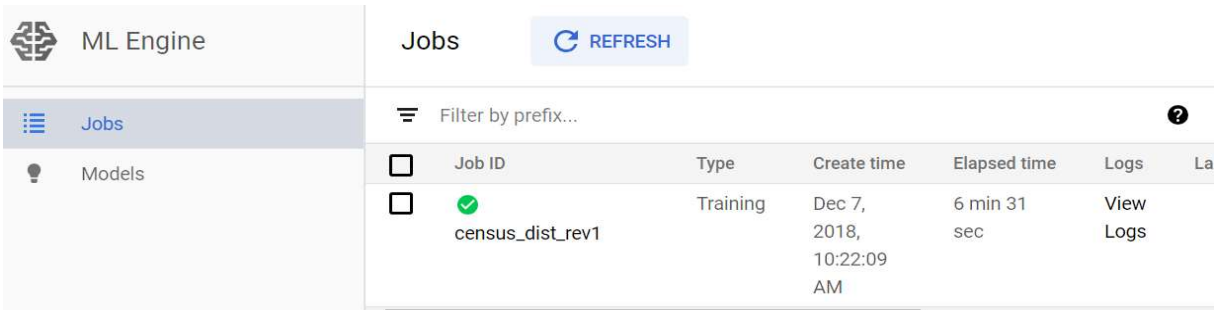
Once the job is submitted correctly, expect to see a message similar to the following:

```
Job [census_dist_rev1] submitted successfully.
Your job is still active. You may view the status of your job with the command

  $ gcloud ml-engine jobs describe census_dist_rev1

or continue streaming the logs with the command

  $ gcloud ml-engine jobs stream-logs census_dist_rev1
jobId: census_dist_rev1
state: QUEUED
beenish_zia@cloudshell:~/cloudml-samples-master/census/estimator (aiguidebzia-rev1)$
```

Monitor job progress by watching the command-line output or in ML Engine > Jobs on the Google Cloud Platform console:



Once complete, expect to see a screen like the following:



View job status on cloud shell with the command:

```
gcloud ml-engine jobs describe census_dist_rev1
```

This generates an output similar to the following screen shot:

```
createTime: '2018-12-07T18:22:09Z'
etag: 5SNGMQy5Zck=
jobId: census_dist_rev1
state: PREPARING
trainingInput:
  args:
  - --train-files
  - gs://aiguidebzia-rev1-mlengine/data/adult.data.csv
  - --eval-files
  - gs://aiguidebzia-rev1-mlengine/data/adult.test.csv
  - --train-steps
  - '1000'
  - --verbosity
  - DEBUG
  - --eval-steps
  - '100'
  jobDir: gs://aiguidebzia-rev1-mlengine/census_dist_rev1
  packageUris:
  - gs://aiguidebzia-rev1-mlengine/census_dist_rev1/packages/e11d736fe9a394e83c3426fb8c7b03541236a197bd96fece9c548947
  pythonModule: trainer.task
  region: us-west1
  runtimeVersion: '1.10'
  scaleTier: STANDARD_1
trainingOutput: {}

View job in the Cloud Console at:
https://console.cloud.google.com/ml/jobs/census_dist_rev1?project=aiguidebzia-rev1

View logs at:
https://console.cloud.google.com/logs?resource=ml.googleapis.com%2Fjob_id%2Fcensus_dist_rev1&project=aiguidebzia-rev1
```

Once the training has completed, the state variable will show "SUCCEEDED"

Step 16: Inspect the logs

There are two ways to inspect the logs generated from the distributed training:

Either go to **GCP Console ML Engine > Jobs** and click **View Logs**, or use the following command on your cloud shell terminal:

```
gcloud ml-engine jobs stream-logs $JOB_NAME
```

Hyperparameter Tuning (Optional)

Hyperparameter tuning helps maximize the model's predictive accuracy. The census example used in this section stores the hyperparameter configuration settings in a YAML file named hptuning_config.yaml. Use this as a template for your specific model and training.

1. Select a new job name and create variables that reference the configuration:

```
HPTUNING_CONFIG=../hptuning_config.yaml
JOB_NAME=cesus_dist_hptune_rev1
echo $BUCKET_NAME
TRAIN_DATA=gs://$BUCKET_NAME/data/adult.data.csv
EVAL_DATA=gs://$BUCKET_NAME/data/adult.test.csv
```

Example screenshot:

```
$ HPTUNING_CONFIG=../hptuning_config.yaml
$ JOB_NAME=cesus_dist_hptune_rev1
$ echo $BUCKET_NAME

$ TRAIN_DATA=gs://$BUCKET_NAME/data/adult.data.csv
$ EVAL_DATA=gs://$BUCKET_NAME/data/adult.test.csv
$
```

2. Set the `OUTPUT_PATH` as done above. Make sure `BUCKET_NAME` is defined and you are not inadvertently using checkpoints between jobs.

   `OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_NAME`

3. Run the following command to submit a training job that uses hyperparameter tuning on multiple nodes:

```
gcloud ml-engine jobs submit training $JOB_NAME \
    --stream-logs \
    --job-dir $OUTPUT_PATH \
    --runtime-version 1.10 \
    --config $HPTUNING_CONFIG \
    --module-name trainer.task \
    --package-path trainer/ \
    --region $REGION \
    --scale-tier STANDARD_1 \
    -- \
    --train-files $TRAIN_DATA \
    --eval-files $EVAL_DATA \
    --train-steps 1000 \
    --verbosity DEBUG  \
    --eval-steps 100
```

4. View the results in GCP console ML Engine > Jobs, similar to what was done before.

Step 17: Deploy a model to support prediction (INFERENCE)

1. Similar to selecting job name at this step, select a model name. The name must start with a letter and can only contain letters, number, and underscores.
   Example:

   `MODEL_NAME=census_rev1`

2. Create a cloud ML engine model:

```
gcloud ml-engine models create $MODEL_NAME --regions=$REGION
```

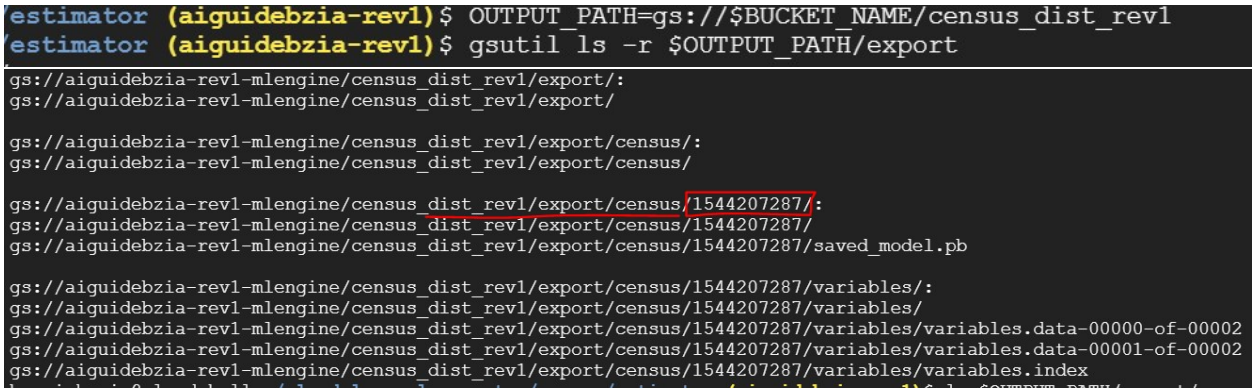
3. Create the output path to use. For example, using **census_dist_rev1** as the job name, which is the same as was created in the distributed non-hyperpararmeter tuning sub-section:

```
OUTPUT_PATH=gs://$BUCKET_NAME/census_dist_rev1
```

4. Find the full path of the exported trained model libraries:

```
gsutil ls -r $OUTPUT_PATH/export
```

Example screenshot:



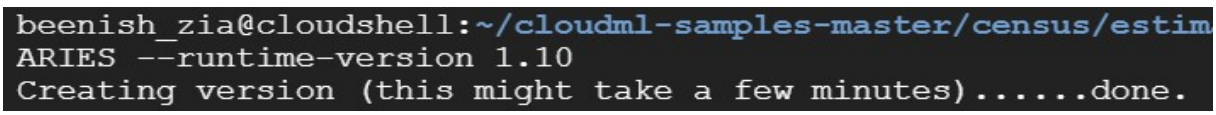
5. In the step above, the screen will show all the directories under `$BUCKET_NAME/export`. Locate directory named `$OUTPUT_PATH/export/census/<timestamp>` and copy this directory path (with the colon ":" at the end) and set that to the `MODEL_BINARIES` label. See the screenshot above with red highlight as an example.

```
MODEL_BINARIES=gs://$BUCKET_NAME/census_dist_rev1/export/census/1
544207287/
```

6. Run the following command to create a version rev1:

```
gcloud ml-engine versions create rev1 --model $MODEL_NAME -origin
$MODEL_BINARIES --runtime-version 1.10
```

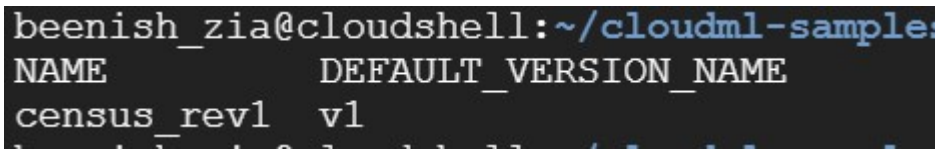
This takes a few minutes. Once complete, expect to see a screen message as shown in the example below.



7. Obtain a list of models using the list command:

```
gcloud ml-engine models list
```

Output result will look like the example below:



Step 18: Send an online prediction request to a deployed model

Now that you've deployed your model, you can send it prediction requests. In the example below, the following command sends an online prediction request using a test.json file that was downloaded as part of the steps above.

```
gcloud ml-engine predict --model $MODEL_NAME --version v1 --json-instances ../test.json
```

This command will result in an output as follows:

```
CLASS_IDS  CLASSES  LOGISTIC              LOGITS                PROBABILITIES
[0]        [u'0']   [0.06204340606927872]  [-2.7158694267272 95]  [0.9379565715789795, 0.06204340606927872]
beenish_zia@cloudshell:~/cloudml-samples-master/census/estimator (aiguidebzia-rev1)$
```

The result indicates whether the predicted income is greater than or less than $50k.

Find details for submitting a batch prediction job in Google Cloud AI and ML Products page. Batch prediction is useful for handling large amounts of data and no latency requirements on receiving prediction results. A run for batch prediction uses the same format as an online prediction, but it uses Cloud Storage for data.

A batch prediction is slower for a small number of instances as it's more suitable for larger data.

Step 19: Cleanup

After analyzing the output from the training and inference run, use the following the command to clean up cloud storage, to avoid incurring additional GCP charges.

```
gsutil rm -r gs://$BUCKET_NAME/$JOB_NAME
```

To learn how to deploy more workloads, Google's AI and Machine Learning Products page has a list of more samples and tutorials .

Conclusion

The goal of this document was to provide beginner level AI practitioners a glimpse of business considerations for AI, with a detailed list of steps to get started deploying TensorFlow on the Intel Xeon platform and run sample training and inference jobs.

Intel, and its many ecosystem partners, can be depended on to provide developer resources to help you get started on your AI journey. Visit Intel® AI to learn more about Intel's rich AI offerings, as well as Intel® AI Builders for an extensive list of AI builder partners, AI blogs, solutions, reference designs, and testimonials.

Appendix

Install the Linux* Operating System

This section requires CentOS-7-x86_64-*1611.iso. This software component can be downloaded from the CentOS website.

DVD ISO was used to implement and verify the steps in this document; you can also use Everything ISO and Minimal ISO.

Steps to Install Linux

1. Insert the CentOS 7.3 1611 install disc/USB. Boot from the drive and select **Install CentOS 7**.

2. Select **Date** and **Time.**

3. If necessary, select **Installation Destination**.

    a. Select the **automatic partitioning** option.

    b. Click **Done** to return home. Accept all defaults for the partitioning wizard, if prompted.

4. Select **Network and hostname**.

    a. Enter "<hostname>" as the hostname.

        i. Click **Apply** for the hostname to take effect.

    b. Select **Ethernet enp3s0f3** and click **Configure** to set up the external interface.

        i. From the **General** section, check **Automatically connect to this network when it's available**.

        ii. Configure the external interface as necessary. **Save** and **Exit.**

    c. Select the toggle to **ON** for the interface.

    d. Select the toggle to **ON** for the interface.

5. Select **Software Selection**. In the box labeled **Base Environment** on the left side, select **Infrastructure server**.

    a. Click **Done** to return home.

    b. Wait until the **Begin Installation** button is available, which may take several minutes. Then click it to continue.

6. While waiting for the installation to finish, set the root password.

7. Click **Reboot** when the installation is complete.

8. Boot from the primary device and log in as root.

References

1. The Many Ways to Define Artificial Intelligence
2. Installing TensorFlow on Ubuntu* (Accessed 6/25/18)
3. Intel® Optimization for TensorFlow* Installation Guide
4. AI Data LifeCycle
5. TensorFlow Download and Setup
6. Horovod Distributed Training on Kubernetes using MLT
7. Installing TensorFlow from Sources
8. A Portable Foreign Function Interface Library (Libffi)
9. Virtualenv
10. Install TensorFlow on CentOS7" (Accessed 6/25/18)
11. The CIFAR-10 dataset
12. CIFAR-10 Details
13. TensorFlow Models
14. Epoch vs Batch Size vs Iterations
15. Learning Multiple Layers of Features from Tiny Images (PDF), Alex Krizhevsky, 2009
16. What is batch size in neural network?
17. Performance Guide for TensorFlow
18. Using Intel® Xeon® for Multi-node Scaling of TensorFlow* with Horovod*
19. Cluster Design Reference Architecture
20. PuTTY download page
21. AI Data LifeCycle
22. Conda Download Guide
23. Install Docker and Learn Basic Container Manipulation in CentOS and RHEL 7/6 – Part 1
24. Custom Docker daemon options
25. TensorFlow Performance Guide - Optimizing for CPU
26. Launch an AWS Deep Learning AMI
27. Git Download Guide
28. How to Install and Use Docker on Ubuntu 16.04
29. Best Known Methods for Scaling Deep Learning with Tensorflow* On Intel® Xeon® Processor Based Clusters
30. What Is the AWS Deep Learning AMI?
31. Getting Started Training Prediction
32. Specifying Machine Types or Scale Tiers
33. Thomas W. Malone, MIT
34. TensorFlow Samples