

Machine Learning

Using GPUs for Cloudera Machine Learning Projects

Date published: 2020-07-16

Date modified: 2022-04-11

CLouDERA

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2023. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Using GPUs for Cloudera Machine Learning projects.....	4
Using GPUs with Legacy Engines.....	4
Custom CUDA-capable Engine Image.....	4
Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment.....	6
Project Admins: Enable the CUDA Engine for your Project.....	6
Testing GPU Setup.....	6
Testing ML Runtime GPU Setup.....	7

Using GPUs for Cloudera Machine Learning projects

A GPU is a specialized processor that can be used to accelerate highly parallelized computationally-intensive workloads. Because of their computational power, GPUs have been found to be particularly well-suited to [deep learning](#) workloads. Ideally, CPUs and GPUs should be used in tandem for data engineering and data science workloads. A typical machine learning workflow involves data preparation, model training, model scoring, and model fitting. You can use existing general-purpose CPUs for each stage of the workflow, and optionally accelerate the math-intensive steps with the selective application of special-purpose GPUs. For example, GPUs allow you to accelerate model fitting using frameworks such as [Tensorflow](#), [PyTorch](#), and [Keras](#).

By enabling GPU support, data scientists can share GPU resources available on Cloudera Machine Learning workspaces. Users can request a specific number of GPU instances, up to the total number available, which are then allocated to the running session or job for the duration of the run.

Enabling GPUs on ML Workspaces

If you are using ML Runtimes, you must use the ML Runtimes version for your Python library Nvidia GPU Edition.



Note: Nvidia GPU Edition comes with CUDA 11.1 preinstalled.

If you are using a Legacy Engine, to enable GPU usage on Cloudera Machine Learning, select GPUs when you are provisioning the workspace. If your existing workspace does not have GPUs provisioned, contact your ML administrator to provision a new one for you. For instructions, see [Provisioning ML Workspaces](#).



Important: Review your cloud service provider account limits

For example, AWS imposes certain default limits for AWS services, and you might not have default access to GPU instances at all. Make sure you review your account's current usage status and resource limits before you start provisioning GPU resources for CML.

Related Information

[Provisioning ML Workspaces](#)

[Custom CUDA-capable Engine Image](#)

[Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment](#)

[Project Admins: Enable the CUDA Engine for your Project](#)

[Testing GPU Setup](#)

Using GPUs with Legacy Engines

To use GPUs with legacy engines, you must create a custom CUDA-capable engine image.

Custom CUDA-capable Engine Image



Note: Before proceeding with creating a custom CUDA-capable engine, the Administrator needs to [install the Nvidia plugin](#).

The base engine image (docker.repository.cloudera.com/CML/engine:<version>) that ships with Cloudera Machine Learning will need to be extended with CUDA libraries to make it possible to use GPUs in jobs and sessions.

The following sample Dockerfile illustrates an engine on top of which machine learning frameworks such as Tensorflow and PyTorch can be used. This Dockerfile uses a deep learning library from NVIDIA called [NVIDIA](#)

[CUDA Deep Neural Network \(cuDNN\)](#). For detailed information about compatibility between NVIDIA driver versions and CUDA, refer the [cuDNN installation guide \(prerequisites\)](#).

When creating the Dockerfile for the custom image, you must delete the Cloudera repository that is inaccessible because of the paywall by running the following:

```
RUN rm /etc/apt/sources.list.d/*
```

Make sure you also check with the machine learning framework that you intend to use in order to know which version of cuDNN is needed. As an example, Tensorflow's NVIDIA hardware and software requirements for GPU support are listed in the [Tensorflow documentation here](#). Additionally, the Tensorflow version compatibility matrix for CUDA and cuDNN is documented [here](#).

The following sample Dockerfile uses NVIDIA's official Dockerfiles for [CUDA and cuDNN images](#).

cuda.Dockerfile

```
FROM docker.repository.cloudera.com/cloudera/cdsw/engine:14-cml-2021.05-1

RUN rm /etc/apt/sources.list.d/*
RUN apt-get update && apt-get install -y --no-install-recommends \
gnupg2 curl ca-certificates && \
curl -fsSL https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7fa2af80.pub | apt-key add - && \
echo "deb https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64 /" > /etc/apt/sources.list.d/cuda.list && \
echo "deb https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64 /" > /etc/apt/sources.list.d/nvidia-ml.list && \
apt-get purge --autoremove -y curl && \
rm -rf /var/lib/apt/lists/*

ENV CUDA_VERSION 10.1.243
LABEL com.nvidia.cuda.version="${CUDA_VERSION}"

ENV CUDA_PKG_VERSION 10-1=${CUDA_VERSION}-1
RUN apt-get update && apt-get install -y --no-install-recommends \
cuda-cudart-${CUDA_PKG_VERSION} && \
cuda-libraries-${CUDA_PKG_VERSION} && \
ln -s cuda-10.1 /usr/local/cuda && \
rm -rf /var/lib/apt/lists/*

RUN echo "/usr/local/cuda/lib64" >> /etc/ld.so.conf.d/cuda.conf && \
ldconfig

RUN echo "/usr/local/nvidia/lib" >> /etc/ld.so.conf.d/nvidia.conf && \
echo "/usr/local/nvidia/lib64" >> /etc/ld.so.conf.d/nvidia.conf

ENV PATH /usr/local/nvidia/bin:/usr/local/cuda/bin:${PATH}
ENV LD_LIBRARY_PATH /usr/local/nvidia/lib:/usr/local/nvidia/lib64:/usr/local/cuda-10.2/targets/x86_64-linux/lib/

RUN echo "deb http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64 /" > /etc/apt/sources.list.d/nvidia-ml.list

ENV CUDNN_VERSION 7.6.5.32
LABEL com.nvidia.cudnn.version="${CUDNN_VERSION}"

RUN apt-get update && apt-get install -y --no-install-recommends \
libcudnn7=${CUDNN_VERSION}-1+cuda10.1 && \
apt-mark hold libcudnn7 && \
rm -rf /var/lib/apt/lists/*
```

Use the following example command to build the custom engine image using the `cuda.Dockerfile` command:

```
docker build --network host -t <company-registry>/CML-cuda:13 . -f cuda.Dockerfile
```

Push this new engine image to a public Docker registry so that it can be made available for Cloudera Machine Learning workloads. For example:

```
docker push <company-registry>/CML-cuda:13
```

Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment

After you create a custom CUDA-capable engine image, you must add the new engine to Cloudera Machine Learning.

About this task

You must have the Site Administrator role to perform this task.

Procedure

1. Sign in to Cloudera Machine Learning.
2. Click Admin.
3. Go to the Engines tab.
4. Under Engine Images, add the custom CUDA-capable engine image created in the previous step.
This allows project administrators across the deployment to start using this engine in their jobs and sessions.
5. Site administrators can also set a limit on the maximum number of GPUs that can be allocated per session or job. From the Maximum GPUs per Session/Job dropdown, select the maximum number of GPUs that can be used by an engine.
6. Click Update.

Project Admins: Enable the CUDA Engine for your Project

You can make the CUDA-capable engine the default engine for workloads within a particular project.

Before you begin

You must be a Project administrator to specify the default engine used for workloads within a particular project.

Procedure

1. Navigate to your project's Overview page.
2. Click Settings.
3. Go to the Engines tab.
4. Under Engine Image, select the CUDA-capable engine image from the dropdown.

Testing GPU Setup

Use these code samples to test that your GPU setup works with several common deep learning libraries. The specific versions of libraries depend on the particular GPU used and the GPU driver version. You can use this testing for GPU setup using ML Runtimes or Legacy Engines.

1. Go to a project that is using the CUDA engine and click Open Workbench.
2. Launch a new session with GPUs.
3. Run the following command in the workbench command prompt to verify that the driver was installed correctly:

```
! /usr/bin/nvidia-smi
```

4. Use any of the following code samples to confirm that the new engine works with common deep learning libraries.

PyTorch

```
!pip3 install torch==1.4.0
from torch import cuda
assert cuda.is_available()
assert cuda.device_count() > 0
print(cuda.get_device_name(cuda.current_device()))
```



Note: The PyTorch installation requires at least 4 GB of memory.

Tensorflow

```
!pip3 install tensorflow-gpu==2.1.0
from tensorflow.python.client import device_lib
assert 'GPU' in str(device_lib.list_local_devices())
device_lib.list_local_devices()
```

Keras

```
!pip3 install keras
from keras import backend
assert len(backend.tensorflow_backend._get_available_gpus()) > 0
print(backend.tensorflow_backend._get_available_gpus())
```

Testing ML Runtime GPU Setup

You can use the following simple examples to test whether the new ML Runtime is able to leverage GPUs as expected.

1. Go to a project that is using the ML Runtimes NVIDIA GPU edition and click Open Workbench.
2. Launch a new session with GPUs.
3. Run the following command in the workbench command prompt to verify that the driver was installed correctly:

```
! /usr/bin/nvidia-smi
```

4. Use any of the following code samples to confirm that the new engine works with common deep learning libraries.

Pytorch

```
!pip3 install torch==1.4.0
from torch import cuda
assert cuda.is_available()
assert cuda.device_count() > 0
print(cuda.get_device_name(cuda.current_device()))
```

Tensorflow

```
!pip3 install tensorflow-gpu==2.1.0
from tensorflow.python.client import device_lib
assert 'GPU' in str(device_lib.list_local_devices())
device_lib.list_local_devices()
```

Keras

```
!pip3 install keras
from keras import backend
assert len(backend.tensorflow_backend._get_available_gpus()) > 0
print(backend.tensorflow_backend._get_available_gpus())
```