Cloudera Data Science Workbench

Using NVIDIA GPUs for Cloudera Data Science Workbench Projects

Date published: 2020-02-28
Date modified:

CLOUDƎRA

Legal Notice

Contents

Using NVIDIA GPUs for Cloudera Data Science Workbench Projects

A GPU is a specialized processor that can be used to accelerate highly parallelized, computationally-intensive workloads.

Minimum Required Roles: Cloudera Manager Cluster Administrator, CDSW Site Administrator

Note: CDSW 1.9.x supports GPUs using the ML Runtimes with 2021.02 or later versions. These versions are automatically fetched over the internet. Support for air gapped environments will be added in the next CDSW release.

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 Data Science Workbench hosts. Users can request a specific number of GPU instances, up to the total number available on a host, which are then allocated to the running session or job for the duration of the run. Projects can use isolated versions of libraries, and even different CUDA and cuDNN versions via Cloudera Data Science Workbench's extensible engine feature.

Key Points to Note

Cloudera Data Science Workbench only supports CUDA-enabled NVIDIA GPU cards.

- Cloudera Data Science Workbench does not support heterogeneous GPU hardware in a single deployment.
- Cloudera Data Science Workbench does not install or configure the NVIDIA drivers on the Cloudera Data Science Workbench worker hosts. These depend on your GPU hardware and will have to be installed by your system administrator. The steps provided in this topic are generic guidelines that will help you evaluate your setup.
- The instructions described in this topic require Internet access. If you have an air-gapped deployment, you will be required to manually download and load the resources onto your hosts.
- Cloudera Data Science Workbench 1.9.0 or later provides two options for supporting GPUs:

  - Support for Nvidia was introduced with ML Runtimes 2021.02. Airgapped environments will have access to ML Runtimes 2021.02 in the upcoming CDSW 1.10 release. See the documentation on the *ML Runtimes Nvidia GPU Edition*.
  - Cloudera Data Science Workbench still provides technical preview support for CUDA-enabled engines. However, CDSW does not include an engine image that supports Nvidia libraries. You must create your own custom CUDA-capable engine image using the instructions provided in *Create a Custom CUDA-capable Engine Image*.

    Note: CUDA-enabled engines will be deprecated with CDSW 1.10 and we recommend using ML Runtimes for GPU support.

- For a list of known issues associated with this feature, refer Known Issues - GPU Support and ML Runtimes Release Notes
- GPU nodes cannot be split among different workloads such as user sessions, jobs, experiments, or models. For example, a 10 GPU node can support 10 different 1-GPU workloads (assuming it can support the CPU and memory requirements). However, the workload must be scheduled completely on the GPU node; it cannot be on a CPU enabled node and then borrow a GPU from a GPU node when needed.

- When scheduling normal workloads, GPU nodes are de-prioritized.

  Cloudera Data Science Workbench uses the following order of preference when scheduling non-GPU workloads (session, job, experiment, or model):

  Worker Hosts > Master Host > GPU-equipped Hosts | Labeled Auxiliary Hosts

  If RESERVE_MASTER is set to true, then the master host is not available for scheduling, so the order of preference becomes:

  Worker Hosts > GPU-equipped Hosts | Labeled Auxiliary Hosts

  When selecting a host to schedule an engine, Cloudera Data Science Workbench gives first preference to unlabeled Worker hosts. If Workers are unavailable or at capacity, CDSW leverages the Master host. And finally, any GPU-equipped hosts OR labeled auxiliary hosts are leveraged.

  GPU-equipped hosts are labeled auxiliary by default so as to reserve them for GPU-intensive workloads. They do not need to be explicitly configured to be labeled. A GPU-equipped host and a labeled auxiliary host are given equal priority when scheduling workloads.

Enabling Cloudera Data Science Workbench to use GPUs

To enable GPU usage on Cloudera Data Science Workbench, perform the following steps to provision the Cloudera Data Science Workbench hosts. As noted in the following instructions, certain steps must be repeated on all gateway hosts that have GPU hardware installed on them.
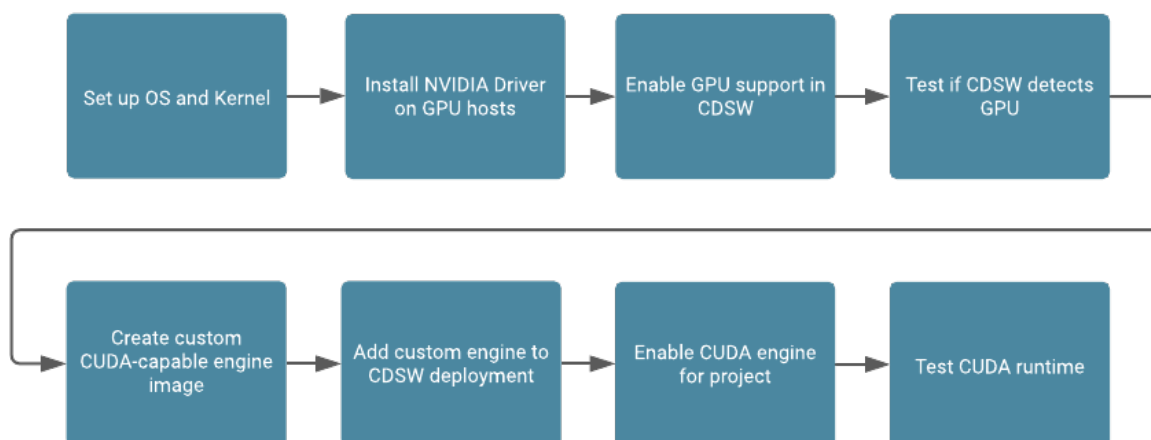
Note: Cloudera recommends using the NVIDIA version of ML Runtimes rather than Legacy Engines with GPUs.

The steps described in this document have been tested and validated on the following setup:

CDSW	OS & Kernel	NVIDIA Driver	CUDA
1.10.x (CUDA-engine:14)	Redhat79 (g3.4xlarge)	NVIDIA driver NVIDIA-Linux-x86_64-470.94.run	CUDA 11.4.1

For more compatibility information across NVIDIA Drivers and CUDA, refer the NVIDIA documentation: CUDA Compatibility.

Set Up the Operating System and Kernel

The first step in enabling GPU usage on Cloudera Data Science Workbench is to set up the operating system and kernel.

Before you begin

Perform this step on all hosts with GPU hardware installed on them.

Procedure

1. Install the kernel-devel package.

```
sudo yum install -y kernel-devel-`uname -r`
```

If the previous command fails to find a matching version of the kernel-devel package, list all the kernel/kernel-devel versions that are available from the RHEL/CentOS package repositories, and pick the desired version to install.

You can use a bash script as demonstrated here to do this:

```
if ! yum install kernel-devel-`uname -r`; then
  yum install -y kernel kernel-devel; retValue=$?
  if [ $retValue -eq 0]; then echo "Reboot is required since new version o
f kernel was installed"; fi
fi
```

2. If you upgraded to a new kernel version in the previous step, run the following command to reboot.

```
sudo reboot
```

3. Install the Development tools package.

```
sudo yum groupinstall -y "Development tools"
```

Install the NVIDIA Driver on GPU Hosts

Cloudera Data Science Workbench does not ship with any of the NVIDIA drivers needed to enable GPUs for general purpose processing. System administrators are expected to install the version of the drivers that are compatible with the CUDA libraries that will be consumed on each host.

About this task

Perform this step on all hosts with GPU hardware installed on them.

Procedure

1. Stop the CDSW service. Login to Cloudera Manager, navigate to the CDSW service, and select Actions Stop .

The CUDA program actively references the service, so if it is not stopped, the following error might occur during installation: ERROR: An NVIDIA kernel   module 'nvidia-drm' appears to already be loaded in your   kernel.

2. Use the NVIDIA UNIX Driver archive to find out which driver is compatible with your GPU card and operating system.

To download and install the NVIDIA driver, make sure you follow the instructions on the respective driver's download page. . It is crucial that you download the correct version.

For example, if you use the .run file method (Linux 64 bit), you would download and install the driver as follows:

```
wget http://us.download.nvidia.com/.../NVIDIA-Linux-x86_
64-<driver_version>.run
export NVIDIA_DRIVER_VERSION=<driver_version>
chmod 755 ./NVIDIA-Linux-x86_64-$NVIDIA_DRIVER_VERSION.run
./NVIDIA-Linux-x86_64-$NVIDIA_DRIVER_VERSION.run -asq
```

3. Once the installation is complete, run the following command to verify that the driver was installed correctly:

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

4. Cloudera recommends installing the Nvidia Container Toolkit to better leverage GPUs in your system.

   Follow the instructions found on NVIDIA's website. Even without this toolkit installed, most GPU-based workloads will run as expected. However some GPU functionalities, for example, running nvidia-smi within a GPU enabled workload, need this toolkit to be installed.

5. Start CDSW. Login to Cloudera Manager, navigate to the CDSW service, and select Actions Start .

   Although CDSW starts running at this point, it can take additional time (for example, 20 minutes) for all CDSW processes to start running.

Enable GPU Support in Cloudera Data Science Workbench

Depending on your deployment, use one of the following sets of steps to enable Cloudera Data Science Workbench to identify the GPUs installed.

Minimum Required Cloudera Manager Role: Cluster Administrator

CSD Deployments

You can enable GPU support in Cloudera Data Science Workbench with CSD deployment.

Before you begin

Minimum Required Cloudera Manager Role: Cluster Administrator

Procedure

1. Go to the CDSW service in Cloudera Manager. Click Configuration. Search for the following property and enable it:

Enable GPU Support	Use the checkbox to enable GPU support for Cloudera Data Science Workbench workloads. When this property is enabled on a host that is equipped with GPU hardware, the GPU(s) will be available for use by Cloudera Data Science Workbench.

2. Restart the CDSW service in Cloudera Manager.

3. Test whether Cloudera Data Science workbench is detecting GPUs.

RPM Deployments

You can enable GPU support in Cloudera Data Science Workbench with RPM deployment.

Before you begin

Minimum Required Cloudera Manager Role: Cluster Administrator

About this task

Procedure

1. Set the following parameter in /etc/cdsw/config/cdsw.conf on all Cloudera Data Science Workbench hosts. You must make sure that cdsw.conf is consistent across all hosts, irrespective of whether they have GPU hardware installed on them.

NVIDIA_GPU_ENABLE	Set this property to true to enable GPU support for Cloudera Data Science Workbench workloads. When this property is enabled on a host that is equipped with GPU hardware, the GPU(s) will be available for use by Cloudera Data Science Workbench.

2. On the master host, run the following command to restart Cloudera Data Science Workbench.

```
cdsw restart
```

If you modified cdsw.conf on a worker host, run the following commands to make sure the changes go into effect:

```
cdsw stop
```

```
cdsw join
```

3. Use the following section to test whether Cloudera Data Science Workbench can now detect GPUs.

Test whether Cloudera Data Science Workbench can Detect GPUs

Once Cloudera Data Science Workbench has successfully restarted, if NVIDIA drivers have been installed on the Cloudera Data Science Workbench hosts, Cloudera Data Science Workbench will now be able to detect the GPUs available on its hosts.

Minimum Required Cloudera Manager Role: Cluster Administrator

Once Cloudera Data Science Workbench has successfully restarted, if NVIDIA drivers have been installed on the Cloudera Data Science Workbench hosts, Cloudera Data Science Workbench will now be able to detect the GPUs available on its hosts.
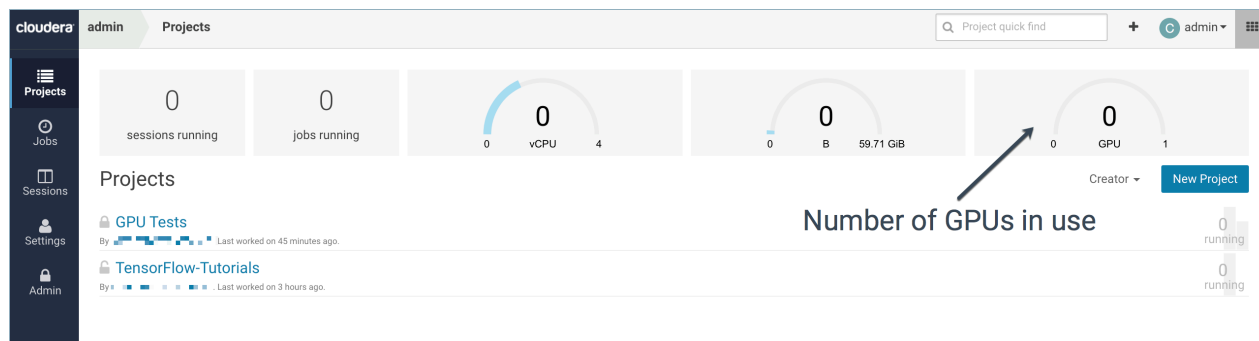
Additionally, the output of this command will also indicate that there are hosts with GPUs present.

```
cdsw status
```

Using GPUs with Legacy Engines-Technical Preview

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

Create a Custom CUDA-capable Engine Image

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

Note: Before you proceed, review the list of known issues and limitations associated with custom engines here.

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, 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.

The following sample Dockerfile uses NVIDIA's official Dockerfiles for CUDA and cuDNN images.

cuda.Dockerfile

```
FROM docker.repository.cloudera.com/cdsw/engine:14
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/ubuntu18
04/x86_64/7fa2af80.pub | apt-key add - && \
echo "deb https://developer.download.nvidia.com/compute/cuda/repos/ubuntu180
4/x86_64 /" > /etc/apt/sources.list.d/cuda.list && \
echo "deb https://developer.download.nvidia.com/compute/machine-learning/re
pos/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 && \
    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

RUN echo "deb http://developer.download.nvidia.com/compute/machine-learning/
repos/ubuntu1604/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/*
```

You can now build a custom engine image out of cuda.Dockerfile using the following sample command:

```
docker build --network host -t <company-registry>/cdsw-cuda:10 . -f cuda.Doc
kerfile
```

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

```
docker push <company-registry>/cdsw-cuda:10
```

Site Admins: Add the Custom CUDA Engine to your Cloudera Data Science Workbench Deployment

After you've created the custom CUDA engine, a site administrator must add this new engine to Cloudera Data Science Workbench.

Before you begin
Required CDSW Role: Site Administrator

Procedure

1. Sign in to Cloudera Data Science Workbench.
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

Project administrators can use the following steps to make it the CUDA engine 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.

Test the CUDA Runtime

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

Procedure

1. Go to a project that is using the CUDA ML Runtime and click New Session.
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
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 Nvdia 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())
```