Cloudera AI

ML Runtimes

Date published: 2020-07-16
Date modified: 2025-10-31

https://docs.cloudera.com/

Legal Notice

Contents

ML Runtimes Overview

Provides overview, installation, set up, configuration, and customization information for ML Runtimes.

ML Runtimes are responsible for running the code written by users and intermediating access to the Cloudera Data Hub.

You can think of an ML Runtime as a virtual machine, customized to have all the necessary dependencies to access the computing cluster while keeping each project's environment entirely isolated. To ensure that every ML Runtime has access to the parcels and client configuration managed by the Cloudera Manager Agent, a number of folders are mounted from the host into the container environment.

ML Runtimes have been open sourced and are available in the cloudera/ml-runtimes GitHub repository. If you need to understand your Runtime environments fully or want to build a new Runtime from scratch, you can access the Dockerfiles that were used to build the ML Runtime container images in this repository.

Using PBJ Workbench

PBJ Workbench offers a classic workbench user interface powered by the open-source Jupyter protocol prepackaged within a runtime image. Users can select this runtime image when launching a session. The open-source Jupyter infrastructure eliminates the dependency on proprietary Cloudera AI code for building a Docker images, enabling faster creation of runtime images. PBJ Workbench enables you to build runtime images using Ubuntu base images, including non-Cloudera base images, and integrate them with the Cloudera AI Workbench.

About this task

ML Runtimes have been open-sourced and are available in the cloudera/ml-runtimes GitHub repository. To gain a deeper understanding of the Runtime environment or to build a new Runtime from scratch, you can access the Dockerfiles used to build the ML Runtime container images in this repository.

The PBJ Workbench is available by default, but you have to select it when you launch a session.

Procedure

1. In the Cloudera console, click the Cloudera AI tile.

   The **Cloudera AI Workbenches** page displays.
2. Click on the name of the workbench.

   The workbench **Home** page displays.
3. Select the required workbench.
4. Select the required project or create a new project.
5. Click the New Session button.
6. In Runtime Editor , select PBJ Workbench
7. Click Start Session.

Results

Now you can use the PBJ Workbench as you would the normal workbench.

Requirements for using a PBJ Workbench

Learn about the prerequisites and preparation steps for setting up a PBJ Workbench.

PBJ Workbench setup: Python installation

PBJ Runtimes must have Python installed, even if the Runtime is designed to run another kernel in Cloudera AI, for example, R kernel. The minimum supported Python version is 3.7. Python can be installed by using the package manager of the base image or can be compiled by the user.

The custom PBJ Runtime image must meet the following essential requirements:

- The actual Python binary or a symlink file pointing to the custom PBJ Runtime image must be located at the following path: /usr/local/bin/python3.
- The Python binary must be included in the PATH environment variable under the *python* name, ensuring that executing the python command in a terminal successfully launches Python.
- Executing python --version must return the result of a Python version higher than version 3.7.
- If the Runtime is configured to run a Python kernel in Cloudera AI, both the python and the /usr/local/bin/python3 commands must launch the same Python process that is registered as a Jupyter kernel.

If the chosen method for installing Python does not place the Python binary under /usr/local/bin/python3, or does not create the python command, create the appropriate symlink files.

Installing Jupyter dependencies and registering your kernel

1. Install the Jupyter kernel Gateway 2.5.2 version into the Docker image.

   You might need to modify this example command depending on the filename and path of the pip executable in the image.

```
RUN pip3 install "jupyter-kernel-gateway==2.5.2"
```

2. Ensure you document the path to the Jupyter executable file installed by the pip package manager. Incorporate the command to run Jupyter kernel Gateway into the ML_RUNTIME_JUPYTER_KERNEL_GATEWAY_CMD environment variable within the Docker image:

```
ENV ML_RUNTIME_JUPYTER_KERNEL_GATEWAY_CMD="/path/to/jupyter kernelgateway"
```

   Note: The highest supported version of the jupyter-client library is version 7.4.9. Do not upgrade to version 8 or higher versions.

   When launching the Runtime in Cloudera AI, the correct IP address, port configuration for Jupyter kernel Gateway is set automatically by Cloudera AI.

3. Register the Jupyter kernel.

   Each instance of the PBJ Workbench communicates with the Jupyter kernel installed in the Runtime image by using the Jupyter protocol. Kernels are available for a wide variety of languages and versions. Install the kernel of your choice to the image by following its installation instructions. A kernel named python3 is registered by default when installing jupyter-kernel-gateway using pip package manager. Installed Jupyter Kernels can be listed by running the following command in a container created from the image:

```
path/to/jupyter kernelspec list
```

4. Defne the name of your chosen kernel within the ML_RUNTIME_JUPYTER_KERNEL_NAME environment variable in the Docker image.

   For example, if the name of your kernel is python3, include the following in the Dockerfile:

```
ENV ML_RUNTIME_JUPYTER_KERNEL_NAME=python3
```

Adding the cdsw user

The user code executes in the image under the user and group identified as 8536:8536. Associate these IDs with the cdsw name in the image by adding the following command to the dockerfile:

```
RUN groupadd --gid 8536 cdsw && \
    useradd -c "CDSW User" --uid 8536 -g cdsw -m -s /bin/bash cdsw
```

Configuring permissions to enable writing Cloudera user settings

All code within the runtime container, including initial setup, executes under the cdsw user. The initial setup includes linking client files for Cloudera Data Services on premises to their standard paths. To enable this process, ensure that the following paths, along with their subfolders, have write permissions for the user ID 8536:

- /etc
- /bin
- /usr/share/java
- /opt
- /usr

Additionally, set the permissions for the following directories, along with all their subdirectories to 777.

- /etc
- /etc/alternatives

Additional requirements

- ML_RUNTIME_METADATA_VERSION environment variable and the corresponding Docker label must be set to value 2.
- To use the PBJ Workbench editor, the ML_RUNTIME_EDITOR environment variable and the corresponding Docker label must be set to PBJ Workbench. If using a 3rd party editor, for example, JupyterLab or RStudio, set the ML_RUNTIME_EDITOR environment variable and the Docker label to the desired value.

  Note: *Workbench* and *PBJ Workbench* are reserved names.

- The base image must be Ubuntu.
- The Bash tool must be installed and must be configured as the default terminal used by the cdsw user.
- When the PBJ Runtime is running the R kernel, the kernel must be registered with the IRkernel package and the bracketed paste mode must be disabled for the bash tool.
- The executable, that is registered as a Jupyter kernel, must be on the PATH environment variable, must be found by the which command and must be named after the programming language of the kernel. For example, the name of the executable must be:
  - python in case of a Python kernel.

    Note: python3 is not a sufficient naming.

  - R in case of an R kernel
- When using a virtual or Conda environment and a Python kernel, Cloudera recommends configuring the PATH environment variable so, that the default pip command corresponds to the Python executable registered as the Jupyter kernel.
- Cloudera AI mounts the project's filesystem under the path /home/cdsw and overwrites any files placed in that location within the Runtime image. Therefore, custom Runtime images must avoid installing any files or configurations under the home folder of the cdsw user.
- Once the Runtime image starts up in Cloudera AI, the kernel must be configured to install new packages to user site libraries under /home/cdsw. That way, newly installed packages persist in the project filesystem.

- The xz-utils package must be installed on the Runtime image.
- The following binaries must be accessible on the PATH variable: `kinit`, `klist`, `ktutil`, and `sshd`. The binaries are installed on Ubuntu as part of the following packages: `krb5-user` and `ssh`.

Dockerfile compatible with PBJ Workbench

This Dockerfile produces a runtime image that is compatible with the PBJ Workbench from a third-party base image.

Note: Apple silicon users should change the first line in the script to the following:

FROM --platform=linux/amd64 ubuntu:22.04

```
FROM ubuntu:22.04
USER root
# Install Python
# Note that the package python-is-python3 will alias python3 as python
RUN apt-get update && apt-get install -y --no-install-recommends \
   krb5-user python3.10 python3-pip python-is-python3 ssh xz-utils

# Configure pip to install packages under /usr/local
# when building the Runtime image
RUN pip3 config set install.user false

# Install the Jupyter kernel gateway.
# The IPython kernel is automatically installed
# under the name python3,
# so below we set the kernel name to python3.
RUN pip3 install "jupyter-kernel-gateway==2.5.2"

# Associate uid and gid 8536 with username cdsw
RUN \
  addgroup --gid 8536 cdsw && \
  adduser --disabled-password --gecos "CDSW User" --uid 8536 --gid 8536 cdsw

# Set up Python symlink to /usr/local/bin/python3
RUN ln -s $(which python) /usr/local/bin/python3

# Install any additional packages.
# apt-get install ...
# pip install ...

# configure pip to install packages to /home/cdsw
# once the Runtime image is loaded into Cloudera AI
# do not install Python packages in the Dockerfile after this line
RUN /bin/bash -c "echo -e '[install]\nuser = true'" > /etc/pip.conf

# Relax permissions to facilitate installation of Cloudera
# client files at startup
RUN for i in /bin /opt /usr /usr/share/java; do \
   mkdir -p ${i}; \
   chown cdsw ${i}; \
   chmod +rw ${i}; \
   for subfolder in `find ${i} -type d` ; do \
      chown cdsw ${subfolder}; \
      chmod +rw ${subfolder}; \
   done \
 done

RUN for i in /etc /etc/alternatives; do \
  mkdir -p ${i}; \
  chmod 777 ${i}; \
  done
```

```
# Set Runtime label and environment variables metadata
#ML_RUNTIME_EDITOR and ML_RUNTIME_METADATA_VERSION must not be changed.
ENV ML_RUNTIME_EDITOR="PBJ Workbench" \
    ML_RUNTIME_METADATA_VERSION="2" \
    ML_RUNTIME_KERNEL="Python 3.10" \
    ML_RUNTIME_EDITION="Custom Edition" \
    ML_RUNTIME_SHORT_VERSION="1.0" \
    ML_RUNTIME_MAINTENANCE_VERSION="1" \
    ML_RUNTIME_JUPYTER_KERNEL_GATEWAY_CMD="/usr/local/bin/jupyter kernelgate
way" \
    ML_RUNTIME_JUPYTER_KERNEL_NAME="python3" \
    ML_RUNTIME_DESCRIPTION="My first Custom PBJ Runtime"


ENV ML_RUNTIME_FULL_VERSION="$ML_RUNTIME_SHORT_VERSION.$ML_RUNTIME_MAINTE
NANCE_VERSION"
LABEL com.cloudera.ml.runtime.editor=$ML_RUNTIME_EDITOR \
      com.cloudera.ml.runtime.kernel=$ML_RUNTIME_KERNEL \
      com.cloudera.ml.runtime.edition=$ML_RUNTIME_EDITION \
      com.cloudera.ml.runtime.full-version=$ML_RUNTIME_FULL_VERSION \
      com.cloudera.ml.runtime.short-version=$ML_RUNTIME_SHORT_VERSION \
      com.cloudera.ml.runtime.maintenance-version=$ML_RUNTIME_MAINTENANCE
_VERSION \
      com.cloudera.ml.runtime.description=$ML_RUNTIME_DESCRIPTION \
      com.cloudera.ml.runtime.runtime-metadata-version=$ML_RUNTIME_METADA
TA_VERSION
```

PBJ Runtimes and Models

The PBJ (Powered by Jupyter) Runtime enables a wide variety of language kernels to be run as Cloudera AI workloads. Model workloads are currently only supported for Python and R kernels.

A new library, cml, is added to Runtime based Python and R workloads automatically and includes mostly the same functionality as the old cdsw library.

In non-PBJ Runtimes, you could point to a Python or R function and run it as a model. PBJ Runtimes open up the possibilities for users to create their own Runtimes and places few restrictions on how to do that. PBJ technology utilizes the Jupyter kernelgateway to communicate with language kernels and aims to be language independent in the future.

Migrating from the cdsw to Cloudera AI library

PBJ Runtimes currently includes two supported languages, Python and R. There are new function wrappers or decorators, described below, that must be used in order to allow previously written model code to function in a PBJ Runtime. It is important to note that these support decorators or wrappers are transparent, that is, they have no effect, in non-PBJ workloads and non-model workloads.

Table 1: Python names

CDSW	Cloudera AI
call_model	models_v1.call_model
-	models_v1.cml_model
get_auth	utils_v1.get_auth
track_file	Removed
model_metrics	Replaced by models_v1.cml_model(metrics=True)

CDSW	Cloudera AI
read_metrics	metrics_v1.read_metrics
track_metric	metrics_v1.track_metric
track_delayed_metrics	metrics_v1.track_delayed_metrics
track_aggregate_metrics	metrics_v1.track_aggregate_metrics
launch_workers	workers_v1.launch_workers
list_workers	workers_v1.list_workers
stop_workers	workers_v1.stop_workers
await_workers	workers_v1.await_workers

Table 2: R names

CDSW	Cloudera AI
html	Planned for future release
iframe	Planned for future release
markdown	Planned for future release
display.help	Planned for future release
code	Planned for future release
image	Planned for future release
get.auth	get.auth
launch.workers	launch.workers
stop.workers	stop.workers
list.workers	list.workers
await.workers	await.workers
track.metric	Removed
track.file	Removed
-	cml_model

Library availability

In Cloudera AI, the cdsw and s libraries are available in different runtime and engine types.

Runtime / Engine type	Library availablility
Legacy Engine	cdsw
Classic Runtime	cdsw, cml
PBJ Runtime	cml

Note: The cml library is now used by default, instead of the cdsw library. However, if a project was created on a previous Cloudera AI release, it might use the deprecated cdsw library. Code using the cdsw library will not run on a PBJ Runtime. Code running on PBJ Runtimes must use the cml library.

Example models with PBJ Runtimes

The library cml includes the package, models_v1. This package includes the cml_model decorator that can be used to allow a function to work as a model in a PBJ Runtime. It can also be used to enable gathering of model metrics.

The package has an optional boolean argument, called metrics, that when set to True, enables the model_metric decorator functionality from the deprecated cdsw lib. The default value of the metrics parameter is False. Therefore, this example does not have model metric gathering enabled, but the R example does. In both examples the function to be used for the model is called predict.

Python example

Example #1

```
import cml.models_v1 as models

@models.cml_model
def predict(args):
    return args["x"]*2
```

Example #2:

```
import cml.models_v1 as models

@models.cml_model(metrics=True)
def predict(args):
    return args["x"]*2
```

R example

Almost all of the functionality in the cdsw library for R has also been migrated to the cml library and is available automatically for every R Runtime workload. It includes a new function wrapper, called cml_model, to be used for model entry point functions in PBJ Runtimes. The function to be used for the model in this case is called predict.

```
library(cml)

predict <- cml_model(function(args) {
    return(args$x*2)
    })
```

ML Runtimes NVIDIA GPU Edition

The NVIDIA GPU Edition Runtimes are built on top of NVIDIA CUDA docker images. CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers can dramatically speed up computing applications by harnessing the power of GPUs.

Cloudera AI supports GPUs using particular ML Runtime Editions. NVIDIA GPUs together with related specialized software can be utilized using the NVIDIA GPU Edition.

All the following configurations have been tested and verified with NVIDIA driver version 550.163.01.

Runtime Version	CUDA Version	Kernels	Editors	Base Image
2025.06	CUDA 12.5	• Python 3.9 • Python 3.10 • Python 3.11 • Python 3.12 • Python 3.13	• PBJ Workbench • JupyterLab	Ubuntu 24.04

ML Runtimes inherit the compatibility requirements for NVIDIA CUDA. For compatibility information, visit the following pages:

- NVIDIA/CUDA compatibility matrix
- Tensorflow requirements
- Pytorch requirements

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())
```

ML Runtimes NVIDIA RAPIDS Edition

The RAPIDS Edition Runtimes are built on top of community built RAPIDS docker images. The RAPIDS suite of software libraries gives you the freedom to execute end-to-end data science and analytics pipelines entirely on GPUs. It relies on NVIDIA CUDA primitives for low-level compute optimization, but exposes that GPU parallelism and high-bandwidth memory speed through user-friendly Python interfaces.

Note: RAPIDS require NVIDIA Pascal or better GPUs. You need to use P3 or newer EC2 instances on AWS to meet this requirement.

Visit rapids.ai for more information.

Note: RAPIDS on Spark is not support as Spark3 is not supported on CDSW.

ML Runtimes RAPIDS edition differs from other Runtime Editions in the following ways:

- Python maintenance versions differ from what is being used in the Standard and NVIDIA GPU edition runtimes. These Python kernels are coming from the RAPIDS base image.
- Pre-installed Python packages and package versions differ from what's in Standard and NVIDIA GPU edition runtimes.

RAPIDS Runtimes are an optional extension of the Cloudera ML Runtime distribution. The RAPIDS images are not distributed automatically. Administrators can register them in the Runtime Catalog. The following RAPIDS editions are available for the 2021.04 Runtime version:

RAPIDS and CUDA Version	**Kernel**	**Editor**	**Base Image**	**Docker**
RAPIDS 0.18 CUDA 11.0	Python 3.7	Workbench editor	rapidsai/rapidsai-core:0.18-cuda11.0-base-ubuntu20.04-py3.7	docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-workbench-python3.7-rapids:2021.12.1-b17
RAPIDS 0.18 CUDA 11.0	Python 3.8	Workbench editor	rapidsai/rapidsai-core:0.18-cuda11.0-base-ubuntu20.04-py3.8	docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-workbench-python3.8-rapids:2021.12.1-b17
RAPIDS 0.18 CUDA 11.0	Python 3.7	JupyterLab editor	rapidsai/rapidsai-core:0.18-cuda11.0-base-ubuntu20.04-py3.7 Note: This image is not based on NVIDIA's JupyterLab installation so RAPIDS library examples are not installed.	docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-jupyterlab-python3.7-rapids:2021.12.1-b17
RAPIDS 0.18 CUDA 11.0	Python 3.8	JupyterLab editor	rapidsai/rapidsai-core:0.18-cuda11.0-base-ubuntu20.04-py3.8 Note: This image is not based on NVIDIA's JupyterLab installation so RAPIDS library examples are not installed.	docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-jupyterlab-python3.8-rapids:2021.12.1-b17

RAPIDS Python environment

The RAPIDS python libraries are distributed and present as a conda environment in the docker image. While it's possible to install additional libraries for your project via conda, these packages will be installed to a non-persistent part of the file system and won't persist between sessions and jobs.

Installing additional libraries via pip is supported and works the same way as in other Runtimes.

Note: The RAPIDS images have the official Anaconda channels configured for conda, if you decide to use conda to install packages, make sure you are familiar with Anaconda's license requirements.

Related Information

Testing ML Runtime GPU Setup

Using Editors for ML Runtimes

Cloudera Machine Learning provides a Workbench UI to edit and run code, but we also provide a preconfigured JupyterLab runtime to allow this. Choose the editor you prefer when launching a ML Runtime session.

Using JupyterLab with ML Runtimes

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data.

You can use JupyterLab to configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab allows you to use extensions that add new components and integrate with existing ones.

You must install any files necessary for JupyterLab, including configuration, customization, extensions, and kernels, into /home/cdsw for each project.

Code Completion in JupyterLab

You can use JupyterLab to write and run code in notebooks or in a traditional .py file. Completion works out of the box in notebooks and consoles. If you are working in a .py file in the text editor, you must create a console to provide the completion suggestions. To do this:

1. Open the .py file.
2. Right click on the document and choose "Create console for editor".
3. Choose the option appropriate to the type of completion you want to perform.

   For example, to get completions on "math", you must first "import math" in the console.

Set JupyterLab Timeouts

You can control JupyterLab timeouts with the following environment variables:

```
JUPYTER_KERNEL_TIMEOUT_SECONDS
JUPYTER_SERVER_TIMEOUT_SECONDS
```

Installing a Jupyter extension

Extensions modify the appearance or behaviour of Jupyter applications (including JupyterLab and Jupyter Notebook).

Before you begin

You must install any files necessary for JupyterLab, including configuration, customization, extensions, and kernels, into /home/cdsw for each project.

Note: Any Jupyter Extension that is not specifically designed for JupyterLab should work fine with Jupyter Notebook, which is preinstalled in Legacy Engine engine:14.

About this task

The following example installs the ipython-sql extension. This extension adds a %sql magic command that allows you to communicate directly with any database supported by SQLAlchemy.

Procedure

1. Launch a JupyterLab session.
2. Open a terminal and run the following command:

```
pip install ipython-sql
```

Note: The previous command is only an example for installing ipython-sql.

3. Open a notebook or console and run the following command:

```
%load_ext sql
```

Results

You can now use the %sql magic command to connect to a particular database, and the %%sql magic command to create a cell containing an SQL query to run against that database. See the JupyterLab documentation for additional information on Jupyter magic commands.

Installing a Jupyter kernel

Jupyter kernels are “connections” through which a notebook (or other part of JupyterLab) can talk to a particular interpreter. CDSW includes one kernel in each JupyterLab Runtime: Python 3.6, Python 3.7, or Python 3.8.

Jupyter kernels are “connections” through which a notebook (or other part of JupyterLab) can talk to an interpreter. ML Runtimes come with a Jupyter kernel for Python 3.6, 3.7 or 3.8 preinstalled, but you may also install Jupyter kernels for bash or JavaScript/TypeScript.

bash

The following example describes how to enable a bash kernel in a particular project.

1. Launch a JupyterLab session.
2. Open a terminal window and run the following command:

```
pip install bash_kernel
python -m bash_kernel.install
```

You should now have the option to launch bash notebooks and consoles.

JavaScript and TypeScript

The following example describes how to enable JavaScript and TypeScript kernels in a particular project. .

1. Customize your PATH environment variable:
   a. Navigate to Project Settings/Advanced.
   b. Set PATH to $HOME/node_modules/.bin:$PATH.
2. Launch a JupyterLab session.
3. Open a terminal window and run the following command:

```
npm install tslab
tslab install
```

You should now have the option to launch JavaScript and TypeScript notebooks and console.

Using Conda Runtime

Users now can create their own Python or R Conda environments within their Cloudera AI Projects that they can use in the JupyterLab editor.

Creating Python Conda environment

Environments, installed packages, and kernel specifications are persisted on the Project file system. An example flow for creating a new Python Conda environment is the following:

1. Register the Conda environment with the following command:

```
conda create --name myenv python=3.10
conda activate myenv
conda install ipykernel
ipython kernel install --user --name=myenv
```

2. Restart the session after executing the commands in the terminal and you can create notebooks in the registered Conda environment.

JupyterLab Conda Runtime (Technical Preview)

Note:

This feature is in Technical Preview and not recommended for production deployments. Cloudera recommends that you try this feature in test or development environments.

You might run into some known issues while using JupyterLab Conda Runtime.

Sessions

When starting a Notebook or a Console for a specific environment, the installed packages will be available and the interpreter used to evaluate the contents of the Notebook or Console will be the one installed in the environment. However, the Conda environment is not "activated" in these sessions, therefore commands like !which python will return with the base Python interpreter on the Runtime. The recommended ways to modify a Conda environments or install packages are the following:

- conda commands must be used with the -n or --name argument to specify the environment, for example conda -n myenv install pandas
- When installing packages with pip, use the %pip magic to install packages in the active kernel's environment, for example %pip install pandas

Applications and Jobs

To start an Application or Job, first create a launcher Python script containing the following line: ! source activate <conda_env_name> && python <job / application script.py>

When starting the Application or Job, select the launcher script as the "Script".

Models

Models are currently not supported for the Conda Runtime.

Spark

Spark is not supported in JupyterLab Notebooks and Consoles.

Spark workloads are supported in activated Conda environments in JupyterLab Terminals, or in Jobs or Applications.

The CDSW libraries for Python and R are not available for the Conda Runtimes.

Installing Additional ML Runtimes Packages

ML Runtimes are preloaded with a few common packages and libraries for Python. However, a key feature of Cloudera AI is the ability of different projects to install and use libraries pinned to specific versions, just as you would on your local computer.

Before you begin

Before downloading or using third-party content, you are responsible for reviewing and complying with any applicable license terms and making sure that they are acceptable for your use case.

About this task

Generally, Cloudera recommends you install all required packages locally into your project. This will ensure you have the exact versions you want and that these libraries will not be upgraded when Cloudera upgrades the ML Runtimes image. You only need to install libraries and packages once per project. From then on, they are available to any new ML Runtimes you spawn throughout the lifetime of the project.

You can install additional libraries and packages from the workbench, using either the command prompt or the terminal.

Note:

Cloudera AI does not currently support installation of packages that require root access to the hosts. For such use-cases, you will need to create a new custom runtime that extends the base image with the required packages. For instructions, see Customized Runtimes on page 23.

About this task

(Python) Install Packages Using Workbench Command Prompt

To install a package from the command prompt:

1. Navigate to your project's Overview page. Click New Session and launch a session.
2. At the command prompt (see Native Workbench Console and Editor) in the bottom right, enter the command to install the package. Some examples using Python have been provided.

Python 3

```
# Installing from console using ! shell operator and pip3:
!pip3 install beautifulsoup4
# Installing from terminal
pip3 install beautifulsoup4
```

About this task

(Python Only) Using a Requirements File

For a Python project, you can specify a list of the packages you want in a requirements.txt file that lives in your project. The packages can be installed all at once using pip/pip3.

1. Create a new file called requirements.txt file within your project:

   ```
   beautifulsoup4==4.6.0
   seaborn==0.7.1
   ```

2. To install the packages in a Python 3 ML Runtimes, run the following command in the workbench command prompt.

```
!pip3 install -r requirements.txt
```

Restrictions for upgrading R and Python packages

Some R and Python packages should not be upgraded because doing so will break the Workbench UI.

For R Runtimes

Do not upgrade the Cairo or RServe packages in your projects using R Runtimes. If you do, you may be unable to launch sessions, jobs, experiments, or applications.

If these packages are upgraded, you will need to start a new project or to delete the directory .local/lib using the CDSW/Cloudera AI Files UI ("show hidden").

For Python Runtimes

Do not upgrade the ipykernel package or its dependencies (ipython, traitlets, jupyter_client, and tornado) in your project using Python Runtimes. If you do, you may be unable to launch sessions, jobs, experiments, or applications.

If these packages are upgraded, you will need to start a new project or to delete the directory .local/lib using the CDSW/Cloudera AI Files UI ("show hidden").

Python Runtimes in Cloudera AI fail to import the setuptools Python library and can fail installing some Python packages

Python Runtimes in Cloudera AI fail to import the setuptools Python library and therefore can fail installing some Python packages when the library setuptools is present on the Runtime or is installed into the Cloudera AI project with version 60.0.0 or higher.

Python 3.10 Runtimes from the 2023.05 Runtime release ship with a newer version of setuptools, so customers can run into this issue when they are using that Runtime. Also they can run into this issue when they are using Custom Runtimes that has a newer setuptools library version or when they install a new setuptools version into their project (regardless of what Runtime they use).

Workaround: Set the environmental variable SETUPTOOLS_USE_DISTUTILS=stdlib either on a project level under Project Settings -> Advanced or on a workbench level under Site Administration -> Runtime -> Environment variables.

Custom Runtime addons with Cloudera AI

Custom ML Runtimes enable you to create runtimes with your own choice of libraries and applications, but also to further customize existing ML Runtimes with additional configuration files or binaries such as connection drivers, without the effort of creating a new custom ML Runtime.

⚠ **Warning:** Using unsecured or unlicensed libraries can cause security threats. You must diligently add the custom ML Runtime addons. For more information, see *Cloudera's Shared Responsibility Model*.

Custom Runtime addon format requirements

1. The name of the addon must contain only lowercase letters, numbers, underscores and hyphens. Also, it must be no longer than 35 characters. When added to Cloudera AI, the addon will be prefixed with the term custom-addon-.
2. Aat least one path must be mentioned in the uploaded json file.
3. Paths in the metadata file must not end with a forward slash (/) character.

4. Requested paths will be symlinked in the workloads filesystem by the cdsw user, therefore these locations must not exist in the workload file system and the cdsw user must have write access to these locations.

Limitations of using Runtime addons

The following limitations are valid for using custom Runtime addons:

- Injecting Python or R libraries into workloads using custom Runtime addons is not supported. To accomplish this, create a custom Runtime instead. For more information, see *Creating customized ML Runtimes*.
- Misconfigured custom Runtime addons can break user workloads, for example, if metadata refers to paths that do not exist in the uploaded tarball archive.
- You cannot add custom Runtime addons using the UI.
- You can only disable but not update or remove custom Runtime addons.
- Custom Runtime addons are mounted to workloads in read-only mode.

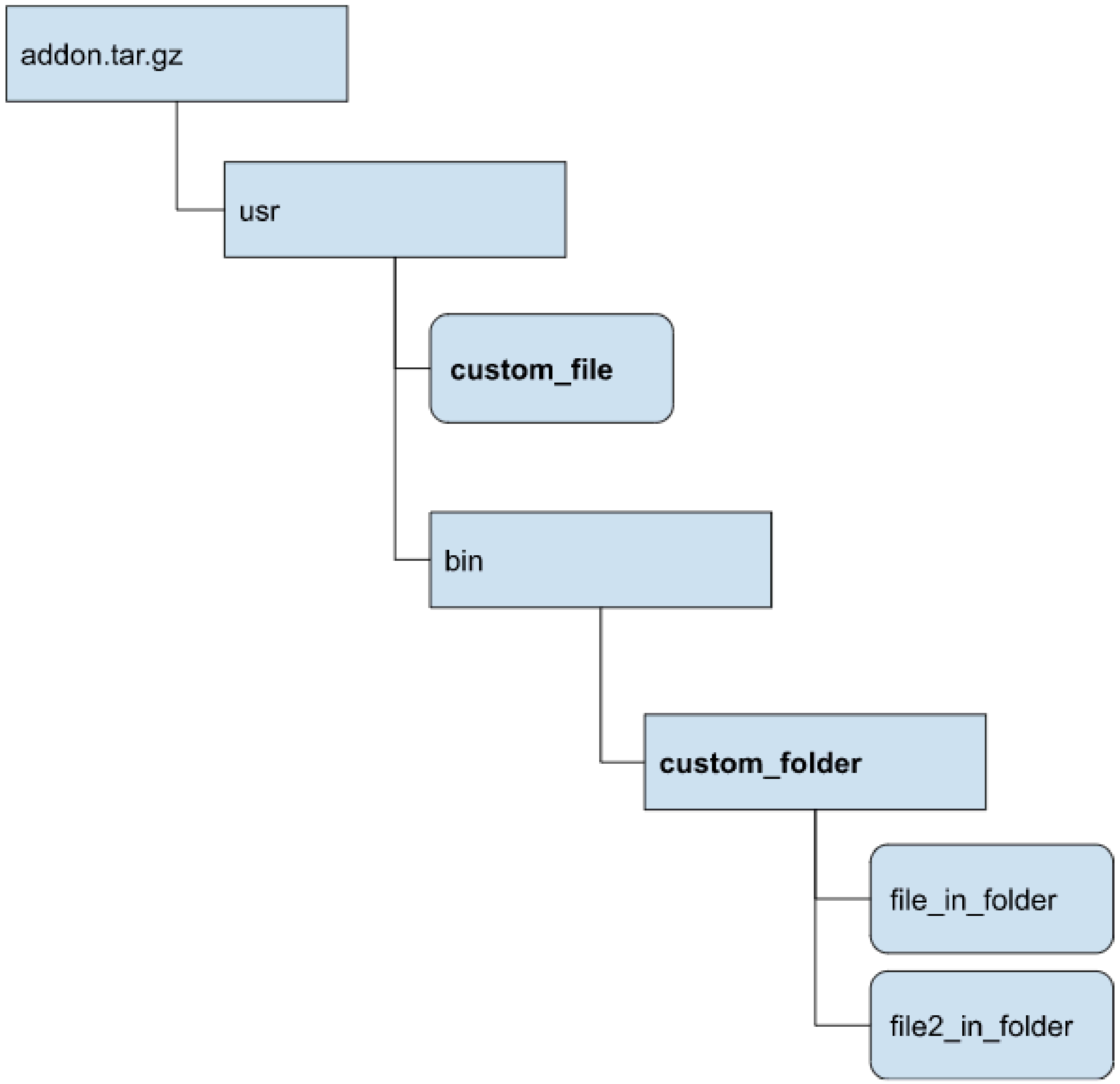
Working with custom Runtime addons

1. Package your dependencies into a tarball file, following the specified format.
2. As an Administrator, register the custom runtime addon by uploading the tarball file using an API call on the command line.
3. After a custom runtime addon is registered, as an Administrator enable or disable addons in the UI.
4. When an addon is enabled, it is mounted to all newly started runtime-based workloads, in read-only mode.

Creating the custom Runtime addon

1. Create a structure of files and folders with the following pattern and package it as a tar.gz archive file.

   For example, to mount a folder called custom_folder to the /usr/bin/custom_folder directory, and the custom_file folder to the /usr/custom_file directory, the tarball file has the following internal structure:



Note:

Ensure that the Runtime Addon file is a valid tar.gz archive. You can verify its validity by using the file addon.tar.gz command.

```
text
        root@ecs1 customRuntime# file addon.tar.gz
        addon.tar.gz: gzip compressed data, from Unix, original size
 modulo 2^32 271134720
```

The output must display gzip compressed data.

2. Create a json file that describes the custom runtime addon to register in Cloudera AI. The json file has the following structure:

```
{
      "name" : "name-of-the-addon",
      "spec" : {"paths" : ["/usr/custom_file", "/usr/bin/custom_folder"]}
      }
```

3. Upload the json and tar.gz files as an Administrator by using the following curl command:

```
curl -v -L '[***ML CLUSTER***]/api/v2/runtimeaddons/custom'  -F metadata=@
metadata.json
      -F tarball=@addon.tar.gz  -H 'Authorization: Bearer [***APIV2
 KEY***]'
```

Substitute the following elements:

- metadata.json: the name of the json file
- addon.tar.gz: the name of the tar.gz file

Once the runtime addon has been uploaded, it will be enabled and mounted to all newly-initiated Runtime-based workloads.

Related Information

Creating Customized ML Runtimes

Troubleshooting custom Runtime addons

Cloudera's Shared Responsibility Model

ML Runtimes Environment Variables

This topic describes how ML Runtimes environmental variables work. It also lists the different scopes at which they can be set and the order of precedence that will be followed in case of conflicts.

ML Runtimes environment variables behave the same way for Legacy Engines and ML Runtimes.

Environmental variables allow you to customize ML Runtimes environments for projects. For example, if you need to configure a particular timezone for a project, or increase the length of the session/job timeout windows, you can use environmental variables to do so.

Cloudera AI allows you to define environmental variables for the following scopes:

Global

A site administrator for your Cloudera AI deployment can set environmental variables on a global level. These values will apply to every project on the deployment.

To set global environmental variables, go to Admin Runtime/Engine .

Project

Project administrators can set project-specific environmental variables to customize the ML Runtimes launched for a project. Variables set here will override the global values set in the site administration panel.

To set environmental variables for a project, go to the project's Overview page and click Settings Advanced .

Job

Environments for individual jobs within a project can be customized while creating the job. Variables set per-job will override the project-level and global settings.

To set environmental variables for a job, go to the job's Overview page and click Settings Set Environmental Variables .

Experiments

ML Runtimes created for execution of experiments are completely isolated from the project. However, these ML Runtimes inherit values from environmental variables set at the project-level and/or global level. Variables set at the project-level will override the global values set in the site administration panel.

Models

Model environments are completely isolated from the project. Environmental variables for these ML Runtimes can be configured during the build stage of the model deployment process. Models will also inherit any environment variables set at the project and global level. However, variables set per-model build will override other settings.

ML Runtimes Environment Variables List

The following table lists Cloudera AI environment variables that you can use to customize your project environments. These can be set either as a site administrator or within the scope of a project or a job.

Environment Variable	Description
MAX_TEXT_LENGTH	It is the maximum number of characters that can be displayed in a single text cell. By default, this value is set to 800,000 and any more characters will be truncated. **Default value:** 800,000
SESSION_MAXIMUM_MINUTES	It is the maximum number of minutes a session can run before it times out. **Default value:** 60*24*7 minutes (7 days) **Maximum value**: 35,000 minutes
JOB_MAXIMUM_MINUTES	It is the maximum number of minutes a job can run before it times out. **Default value:** 60*24*7 minutes (7 days) **Maximum value**: 35,000 minutes
IDLE_MAXIMUM_MINUTES	It is the maximum number of minutes a session can remain idle before it exits. In case of a Jupyterlab session, this is the maximum number of minutes a kernel can remain idle before it exits. **Default value:** 60 minutes **Maximum value**: 35,000 minutes Idle timeouts for sessions vary by workbench type (runtime). • Standard Workbench: Sessions timeout regardless of activity in the browser or terminal. • PBJ Workbench: Sessions timeout if there is no browser activity and no terminal window is open. If a terminal window is open, the session will not timeout, regardless of whether there is activity in the terminal window. • Jupyterlab: Sessions timeout if there is no browser activity. The terminal window activity is not considered. • Custom runtimes: No idle timeout behavior is enforced on custom or third-party workbenches.
JUPYTER_SERVER_TIMEOUT_SECONDS	It is the maximum number of seconds a Jupyterlab session can run without any activity before it exits. Having the Session open in an active browser tab and having running kernels within Jupyterlab count as activity. **Default value:** 300 seconds

Per-Engine Environmental Variables: In addition to the previous table, there are some more built-in environmental variables that are set by the Cloudera AI application itself and do not need to be modified by users. These variables are set per-engine launched by Cloudera AI and only apply within the scope of each engine.

Environment Variable	Description
CDSW_PROJECT	The project to which this engine belongs.
CDSW_ENGINE_ID	The ID of this engine. For sessions, this appears in your browser's URL bar.
CDSW_MASTER_ID	If this engine is a worker, this is the CDSW_ENGINE_ID of its master.
CDSW_MASTER_IP	If this engine is a worker, this is the IP address of its master.
CDSW_PUBLIC_PORT	**Note:** This property is deprecated. See CDSW_APP_PORT and CDSW_READONLY_PORT for alternatives. A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_PUBLIC_PORT will be available in browsers at: http(s)://<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. By default, CDSW_PUBLIC_PORT is set to 8080. A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera AI web application, as long as the job or session is still running. For more details, see *Accessing Web User Interfaces from Cloudera AI*. In Cloudera AI, setting CDSW_PUBLIC_PORT to a non-default port number is not supported.
CDSW_APP_PORT	A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_APP_PORT will be available in browsers at: http(s)://<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. Use this port for applications that grant some control to the project, such as access to the session or terminal. A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera AI web application as long as the job or session runs. Even if the web UI does not have authentication, only Contributors and those with more access to the project can access it. For more details, see *Accessing Web User Interfaces from Cloudera AI*. Note that if the Site Administrator has enabled Allow only session creators to run commands on active sessions, then the UI is only available to the session creator. Other users will not be able to access it. Use 127.0.0.1 as the IP.
CDSW_READONLY_PORT	A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_READONLY_PORT will be available in browsers at: http(s)://<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. Use this port for applications that grant read-only access to project results. A direct link to these web services will be available to users with from the grid icon in the upper right corner of the Cloudera AI web application as long as the job or session runs. Even if the web UI does not have authentication, Viewers and those with more access to the project can access it. For more details, see *Accessing Web User Interfaces from Cloudera AI*. Use 127.0.0.1 as the IP.
CDSW_DOMAIN	The domain on which Cloudera AI is being served. This can be useful for iframing services, as demonstrated in *Accessing Web User Interfaces from Cloudera AI*.
CDSW_CPU_MILLICORES	The number of CPU cores allocated to this engine, expressed in thousandths of a core.
CDSW_MEMORY_MB	The number of megabytes of memory allocated to this engine.
CDSW_IP_ADDRESS	Other engines in the Cloudera AI cluster can contact this engine on this IP address.

Accessing Environmental Variables from Projects

This topic shows you how to access environmental variables from your code.

Environmental variables are injected into every engine launched for a project, contingent on the scope at which the variable was set (global, project, etc.). The following code samples show how to access a sample environment variable called *DATABASE_PASSWORD* from your project code.

Python

```
import os
database_password = os.environ["DATABASE_PASSWORD"]
```

Appending Values to Environment Variables:

You can also set environment variables to append to existing values instead of replacing them. For example, when setting the LD_LIBRARY_PATH variable, you can set the value to LD_LIBRARY_PATH:/path/to/set.

Customized Runtimes

This topic explains how custom Runtimes work and when they should be used.

Note: Cloudera, Inc. ("Cloudera") makes the Custom Runtime feature available to allow its users to add, run, and manage their own container images for their workloads as 'Custom Runtimes'. Cloudera's support covers only the integration points of these images with Cloudera AI, to the extent adding, running, and managing Custom Runtime images fulfill the documented requirements and/or pre-requisites. Cloudera does not provide any support for any other configurations and/or third party software added or installed to the image by customers.

By default, ML Runtimes are preloaded with a few common packages and libraries for R, Python, and Scala. In addition to these, ML Runtimes also allows you to install any other packages or libraries that are required by your projects. However, directly installing a package to a project as described above might not always be feasible. For example, packages that require root access to be installed, or that must be installed to a path outside /home/cdsw (outside the project mount), cannot be installed directly from the workbench.

For such circumstances, Cloudera AI allows you to extend the base Docker image and create a new Docker image with all the libraries and packages you require. Site administrators can then add this new image in the allowlist for use in projects.

Note: You will need to remove any unnecessary Cloudera sources or repositories that are inaccessible because of the paywall.

PBJ Custom Runtimes can be built on top of any Ubuntu base image, and users have to install the kernel themselves. However, non-PBJ Runtime images can only be built on top of Cloudera-released non-PBJ Runtime images, and users cannot change the kernel.

Note that this approach can also be used to accelerate project setup across the deployment. For example, if you want multiple projects on your deployment to have access to some common dependencies (package or software or driver) out of the box, or even if a package just has a complicated setup, it might be easier to simply provide users with a Runtime that has already been customized for their project(s).

Related Resources

- The Cloudera Engineering Blog post on *Customizing Docker Images in Cloudera AI* describes an end-to-end example on how to build and publish a customized Docker image and use it as an engine in Cloudera AI.
- For an example of how to extend the base engine image to include Conda, see *Installing Additional Packages*.

Creating Customized ML Runtimes

This section walks you through the steps required to create your own custom ML Runtimes based on one of the Cloudera provided ML Runtime images.

Note: In case of PBJ Runtimes, the Custom runtime does not have to be built on the top of a Cloudera provided ML Runtime image.

Create a Dockerfile for the Custom Runtime Image

This topic shows you how to create a Dockerfile for a custom image.

First, select an appropriate source image for your customization. For a non-PBJ Runtime, you must use a Runtime image released by Cloudera. Image tags can be seen on the Session Start page on the user interface when you select a Runtime. The second step when building a customized image is to create a Dockerfile that specifies which packages you would like to install in addition to the base image.

For example, the following Dockerfile installs the telnet package, the sklearn Python package and upgraded base packages on top of an ML Runtime image released by Cloudera.

```
# Dockerfile
# Specify an ML Runtime base image
FROM docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-jupyterlab-pyth
on3.7-standard:2021.12.1-b17
# Install telnet in the new image
RUN apt-get update && apt-get install -y --no-install-recommends telnet &&
 apt-get clean && rm -rf /var/lib/apt/lists/*
# Upgrade packages in the base image
RUN apt-get update && apt-get upgrade -y && apt-get clean && rm -rf /var/lib
/apt/lists/*
# Install the python package sklearn
RUN pip install --no-cache-dir sklearn
# Override Runtime label and environment variables metadata
ENV ML_RUNTIME_EDITION="Telnet Edition" \
        ML_RUNTIME_SHORT_VERSION="1.0" \
        ML_RUNTIME_MAINTENANCE_VERSION=1 \
        ML_RUNTIME_DESCRIPTION="This runtime includes telnet and sklearn
and upgraded packages"
ENV ML_RUNTIME_FULL_VERSION="${ML_RUNTIME_SHORT_VERSION}.${ML_RUNTIME_MAI
NTENANCE_VERSION}"
LABEL com.cloudera.ml.runtime.edition=$ML_RUNTIME_EDITION \
        com.cloudera.ml.runtime.full-version=$ML_RUNTIME_FULL_VERSION \
        com.cloudera.ml.runtime.short-version=$ML_RUNTIME_SHORT_VERSION \
        com.cloudera.ml.runtime.maintenance-version=$ML_RUNTIME_MAINTENANCE
_VERSION \
        com.cloudera.ml.runtime.description=$ML_RUNTIME_DESCRIPTION
```

Metadata for Custom ML Runtimes

This topic addresses the metadata for custom Runtimes.

All new custom Runtimes must override the Edition metadata of existing Runtimes. The rest of the metadata can be overriden to communicate the expectations for the consumers of the image. Both the Docker label and the environment variable match in a custom Runtime image. In order to add or register a custom Runtime to a deployment, the user facing metadata combination should be unique in that deployment. For example, of the following, Editor, Edition, Kernel, Version, and Maintenance Version, at least the later should be incremented for adding a next iteration of the same image.

See the following reference table for more details on ML Runtime metadata.

Environment variable	Docker Label	Description	Override in custom non-PBJ runtime	Value in PBJ Runtimes
ML_RUNTIME_METADATA_VERSION	com.cloudera.ml.runtime.runtime-metadata-version	Metadata version	Not allowed	Must be set to 2

ML_RUNTIME_EDITOR	com.cloudera.ml.runtime.editor	CDSW/Cloudera AI Editor installed in the image.	Allowed	Required
ML_RUNTIME_EDITION	com.cloudera.ml.runtime.edition	Edition of the image, a notion of the Runtime capabilities.	Required	Required
ML_RUNTIME_DESCRIPTION	com.cloudera.ml.runtime.description	Longer description of the Runtime image capabilities.	Recommended	Required
ML_RUNTIME_KERNEL	com.cloudera.ml.runtime.kernel	Main kernel included in the image, e.g., Python 3.8	Not allowed	Required
ML_RUNTIME_SHORT_VERSION	com.cloudera.ml.runtime.short-version	Main version of the image, e.g., 1.0. This shows up as Version in the selection screen.	Recommended	Required
ML_RUNTIME_FULL_VERSION	com.cloudera.ml.runtime.full-version	Full version consists of the short version + maintenance version, e.g., 1.0.1	Optional	Optional
ML_RUNTIME_MAINTENANCE_VERSION	com.cloudera.ml.runtime.maintenance-version	Maintenance version must be an integer, e.g., 1. Only the largest maintenance version of the set of the same short version images are visible for users to select. Increment this number if you create a drop in replacement of an existing Runtime. Start with 1 with a new version or edition.	Recommended	Required
ML_RUNTIME_CUDA_VERSION	com.cloudera.ml.runtime.cuda-version	CUDA version installed in the image.	Not allowed	Do not set
ML_RUNTIME_GIT_HASH	com.cloudera.ml.runtime.git-hash	Git hash of runtime source	Not allowed	Do not set
ML_RUNTIME_GBN	com.cloudera.ml.runtime.gbn	Cloudera internal build number	Not allowed	Do not set

Editor Customization

This topic addresses customizing a third-party editor to work with ML Runtimes.

For a third-party editor to work with ML Runtimes, you must provide a starting script. For example:

```
#! /bin/bash
"$ZEPPELIN_HOME/bin/zeppelin.sh"
```

For Cloudera Machine Learning to interpret this as an editor startup script, you must create a symlink to the editor as/usr/local/bin/ml-runtime-editor. This will be created in the Dockerfile of the customized runtime.

Note: Third-party editors provide a way to run arbitrary code that is not distributed by Cloudera. Use third-party editors at your own risk. You should absolutely trust the code that you want to run.

Build the New Docker Image

This topic shows you how to use Docker to build a custom image.

A new custom Docker image can be built on any host where Docker binaries are installed, source images are available, and OS repositories and other package repositories are available. To install these binaries, run the following command on the host where you want to build the new image:

```
docker build -t <image-name>:<tag> . -f Dockerfile
```

If your Dockerfile makes any outside connection (for example, apt-get, update, pip install, curl), you must add the --network=host option to the build command:

```
docker build --network=host -t <image-name>:<tag> . -f Dockerfile
```

Distribute the Image

This topic explains the different methods that can be used to distribute a custom ML Runtime to all the hosts.

Once you have built a new custom ML Runtime, use one of the following methods to distribute the new image to all your Cloudera AI hosts:

Push the image to a public registry such as DockerHub

For instructions, refer to the Docker documentation *docker push* and *Push images to Docker Cloud*.

Push the image to your company's Docker registry

When using this method, make sure to tag your image with the following schema:

```
docker tag <image-name> <company-registry>/<user-name>/<image-na
me>:<tag>
```

Once the image has been tagged properly, use the following command to push the image:

```
docker push <company-registry>/<user-name>/<image-name>:<tag>
```

Adding a new customized ML Runtime through the Runtime Catalog

Cloudera AI enables you to add customized ML Runtimes from the Runtime Catalog window.

About this task

Note: You must have system administrator permission to add a new ML Runtime.

Note: If you add a Custom Runtime from a private docker registry, you need to add the docker credentials first to Cloudera AI. See *Add Docker registry credentials and certificates* for more information.

Procedure

1. Click Runtime Catalog from the Navigation panel.
2. Click the Add Runtime button in the upper right corner.
3. In the Add Runtime window, enter the url of the Runtime Docker image you want to upload.

   As ML Runtimes are identified based on certain attributes, metadata (such as Editor, Kernel, Edition, Version, and Maintenance Version) must be unique to add new Customized Runtimes to a deployment. Customized ML Runtimes must have different Edition text compared to Cloudera supported versions.
4. If you want to use a non-default credential, select a docker credential in the Select Credentials dropdown list.

   The Docker server name and the user name of the credential is displayed.

5. Click Validate.

   Cloudera AI will use the provided URL to fetch the Docker image and validate if it can be used as a customized Runtime.

   If the Docker image is successfully validated, Cloudera AI will display the metadata information of the image. The new customized Runtime will be visible in the Runtime Catalog and accessible over the different workloads.

Related Information

Adding Docker registry credentials

Limitations

This topic lists some limitations associated with customized ML Runtime images.

- The contents of certain pre-existing standard directories such as /home/cdsw, /tmp, and so on, cannot be modified while creating customized non-PBJ ML Runtimes. This means any files saved in these directories will not be accessible from sessions that are running on customized ML Runtimes.

  Workaround: Create a new custom directory in the Dockerfile used to create the customized ML Runtime, and save your files to that directory.

For PBJ Runtimes, note the following limitations:

- PBJ Runtimes work as models only with R and Python kernels.

Adding Docker registry credentials

To enable Cloudera AI to fetch custom ML Runtimes from a secure repository, as an Administrator you need to add Docker registry credentials. If you want to use different credentials for different runtimes, you can add more docker credentials using the UI and API v2 and use that credentials to fetch custom ML Runtimes.

Note: Fetching custom ML Runtimes from registries that require a custom TLS certificate is not supported.

1. In the **Cloudera** console, click the Cloudera AI tile.

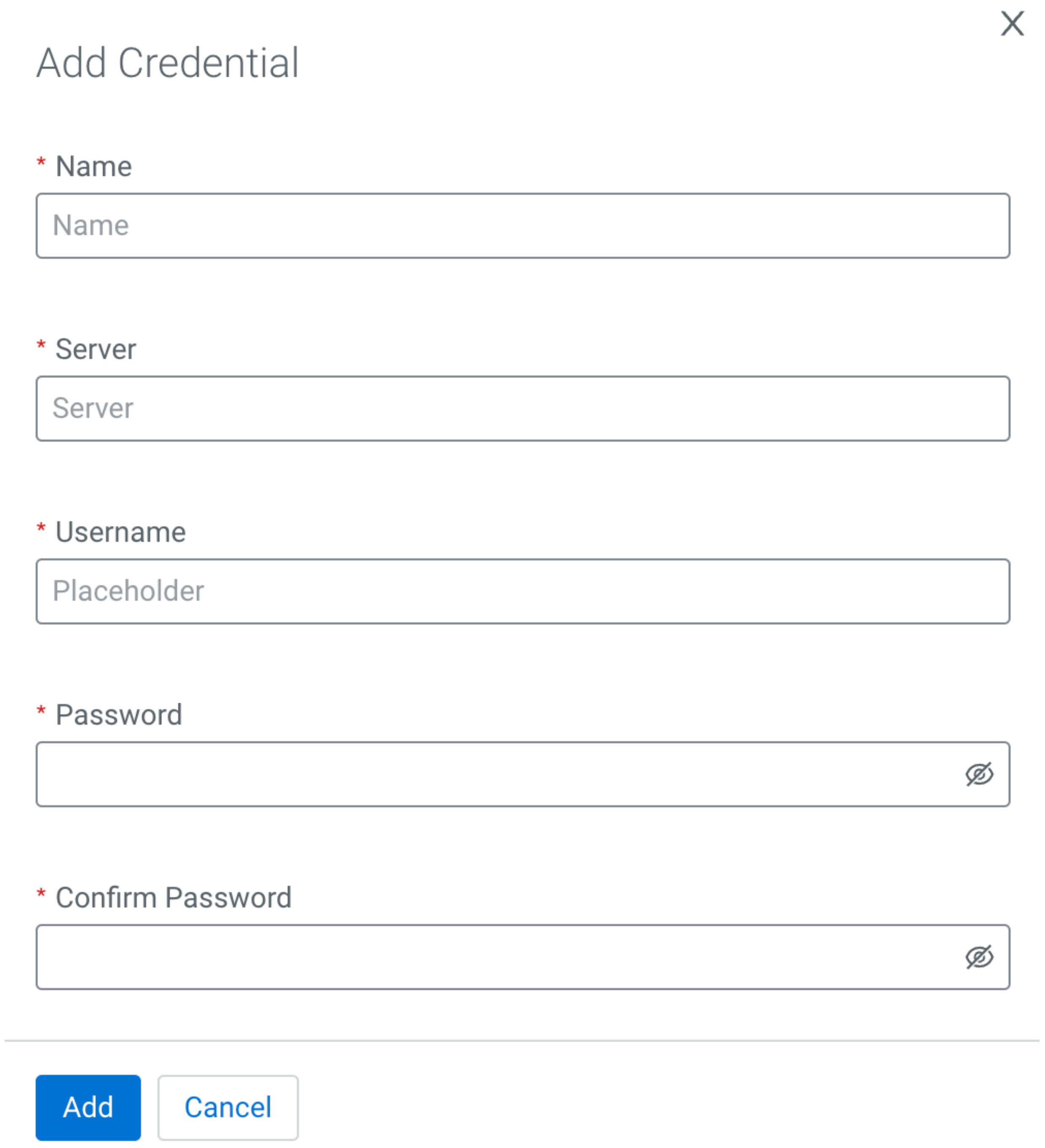
   The **Cloudera AI Workbenches** page displays.
2. Click on the name of the workbench.

   The **Workbenches** Home page displays.
3. Click Site Administration in the left navigation menu.

   The Site Administration page displays.

4. Click Runtimes, and click Add Credentials in the Docker Credentials section.

   The Add Credentials dialog box is displayed.



Add Credential

* Name

Name

* Server

Server

* Username

Placeholder

* Password

* Confirm Password

5. In Name, provide a name for the credential. This name will be displayed to manage the credentials.
6. In Server, provide the docker server address.
7. In Username, provide the username of the credential.
8. In Password, provide the password for the credential.
9. In Confirm Password, repeat the same password given in the Password field.
10. Click Add.

The credentials name and server name are displayed under the Docker Credentials section. The default credentials are used when ML Runtimes images are pulled, and non-default credentials are used when they are added to a Custom Runtime image in **Runtime Catalog**.

Click Add Credentials to add more docker credentials. You can use the toggle button to set the default credential. The default credential is picked up automatically for Custom Runtimes coming from the server stored in the default credential. To use your other credentials, add them to Custom Runtimes in Runtime Catalog.

Click to edit the Docker credential and click to delete the Docker credential.

Note: Refer the documentation of your chosen docker registry to understand what docker login credentials it expects. The expected credentials might be different from your own username and password. Many registries require docker authentication using tokens or API keys.

If the Custom Runtime is uploaded to AWS ECR, then the Docker password would come from your AWS account (aws ecr get-login-password --region <region>). You have to use this password when you add the credential to Cloudera AI.

You need to update these Docker credentials for Cloudera AI again if you back up and then restore your Cloudera AI Workbench.