Cloudera AI

Managing ML Runtimes

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

https://docs.cloudera.com/

Legal Notice

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

Managing ML Runtimes

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.

Adding new ML Runtimes

Cloudera AI provides two ways to add new Runtimes to the Runtime Catalog.

Note: You must have system administrator permission to add a new ML Runtime. Alternatively, in Site Administration Settings Access Control , an admin can enable Allow users to register ML Runtimes. If enabled, the names of users who register ML Runtimes is displayed on the Runtime Catalog page. Users are not permitted to deprecate or disable the added runtimes.

1. Adding Custom ML Runtimes through the Runtime Catalog

   Create your own Custom ML Runtime and add it through the Runtime Catalog
2. Adding ML Runtimes using Runtime Repo files

   Host a file from where Cloudera AI will automatically pull in new ML Runtimes and add them to the Runtime Catalog.

When adding Runtimes from a third party repository, you might need to add Docker credentials and certificates to Cloudera AI. See more on the *Adding docker registry credentials and certificates* page.

Related Information

Adding Docker registry credentials and certificates

Adding Custom ML Runtimes through the Runtime Catalog

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

When you open the Runtime Catalog as an administrator, you can add a new Custom Runtime with the Add Runtime button. You can read more about building a Custom Runtime image and adding to the catalog in the topic Creating Customized ML Runtimes.

Note, that you can not add Cloudera created Runtimes through the Runtime Catalog.

Related Information

Creating customized ML Runtimes

Adding ML Runtimes using Runtime Repo files

Runtime Repo files are JSON files that contain all the details of ML Runtimes that are needed by Cloudera AI to add these ML Runtimes to the Runtime Catalog. When these files are hosted on URLs accessible to Cloudera AI, these URLs can be registered in Cloudera AI. If Enable Runtime Updates is selected in Site Administration Runtime , ML Runtimes that appear in the Runtime Repo files will be automatically added to the Runtime Catalog within 24 hours. Additionally, you can click Update Runtimes now to immediately update the Runtime Catalog. Using Runtime Repo files, both Custom and Cloudera provided Runtimes can be added to Cloudera AI Workbench.

To add, edit, or remove Runtime Repo files

1. Log in as administrator.
2. Navigate to Site Administration Runtime .
3. Add, edit or delete Runtime Repo files in the Runtime Updates section.

Cloudera provided Runtime Repo files

Cloudera AI hosts two public Runtime Repo files that contain the details of the latest released ML Runtimes and Data Visualization Runtimes.

Name: Cloudera ML Runtimes

URL: https://archive.cloudera.com/ml-runtimes/latest/artifacts/repo-assembly.json

Name: Cloudera DataViz Runtimes

URL: https://archive.cloudera.com/cdv/latest/artifacts/repo-assembly.json

These Runtime Repo files are not added to Cloudera AI on premises clusters by default. However, if your cluster is not air gapped, you can add these Runtime Repo files to your Cloudera AI Workbench to keep your Runtime Catalog automatically updated with the latest released Cloudera Runtimes. Note, that these Runtimes will be added to your Runtime Catalog from Cloudera hosted registries, even if your on premises is configured to use a different docker registry.

If your cluster is air-gapped, you can still add new Cloudera-built ML Runtimes to your Cloudera AI Workbench by pushing these Runtimes into a docker registry that is hosted within your air-gapped environment, and then adding a Runtime Repo file to Cloudera AI. The image_identifier fields in the file must point to the self-hosted docker registry instead of any Cloudera-hosted registries.

Self created Runtime Repo files

You can create your own Runtime Repo files and register them in Cloudera AI. Cloudera AI checks these Runtime Repo files for changes every 24 hours and adds any new ML Runtimes found in these files automatically to the Runtime Catalog.

To create a Repo assembly file:

1. Create a JSON file with the same structure as the Cloudera provided one:

```
{
    "assembly_metadata_version": 1,
    "runtimes": [
         {
           "image_identifier": string,
            "runtime_metadata_version": int,
            "editor": string,
            "edition": string,
            "description": string,
            "kernel": string,
            "full_version": string,
            "short_version": string,
            "maintenance_version": int,
            "git_hash": string,
            "gbn": int
         } ,
    ]
}
```

2. Fill in the details of one or more ML Runtimes. If you are adding Cloudera-created Runtimes, use the values from the Cloudera-provided Runtime Repo files. For Custom Runtimes, git_hash should be an empty string, and gbn should be set to zero. All other fields should be filled according to the information in *Metadata for Custom ML Runtimes*.
3. Host the JSON file on an URL that Cloudera AI is able to access.
4. Add the Runtime Repo file to Cloudera AI on the Site Administration Runtime page.

Related Information

Metadata for custom ML Runtimes

Updating ML Runtime images on Cloudera AI installations

In on premises ML Runtimes, the collection of docker images are bundled as part of the package. However, you can upgrade Runtime images to the latest version any time and even if you have an air-gapped installation, that is, there is no or only limited Internet access.

For Airgapped installations (with no or limited Internet access)

⚠ **Important:** You need to be in non-airgapped environment (have Internet access) to carry out some of the tasks.

1. Download ML Runtimes from the locations specified in the repo-assembly.json file:

   https://archive.cloudera.com/ml-runtimes/[***RUNTIMES VERSION***]/artifacts/repo-assembly.json

   ```
   export ml_runtime_version=[***RUNTIME VERSION***]
   wget -O repo-assembly.json
   https://archive.cloudera.com/ml-runtimes/${ml_runtime_version}/artifacts
   /repo-assembly.json
   ```
2. Get the list of images for a particular release, that is ${ml_runtime_version} using the following command:

   ```
   curl
   https://archive.cloudera.com/ml-runtimes/${ml_runtime_version}/artifac
   ts/repo-assembly.json | jq -r '.runtimes[] | .image_identifier'
   ```
3. Ensure that docker is running on your machine or environment.
4. Use the docker pull and docker save commands for each of the images in the .json manifest file to create a tar.gz file, that you can copy to your air-gapped environment.

   ```
   # ensure you have enough space in image_dir
   image_dir=<path to image dir in non-airgapped environment>
   mkdir -p ${image_dir}
   for i in $(jq -r '.runtimes[].image_identifier' repo-assembly.json)
   do
     image=$(echo $i | awk -F'/' ' { printf("%s",$NF)} ')
     image_file="${image}.tar.gz"
     echo "docker pull ${i}"
     docker pull ${i}
     echo "docker save ${i} -o \"${image_dir}/${image_file}\""
     docker save ${i} -o "${image_dir}/${image_file}"
   done
   ```
5. Transfer the tar.gz files to the air-gapped environment.
6. Upload the images to the existing docker registry of the air-gapped environment.
7. Use the docker load, docker tag and docker push commands to save the images to the air-gapped environment.

   ```
   # you will need a copy of the repo-assembly.json used above
   ```

```
remote_image_dir=<path to image dir in airgapped environment>
DOCKER_REGISTRY_DEST=<custom docker repository:port>
for i in $(jq -r '.runtimes[].image_identifier' repo-assembly.json)
do
  image=$(echo $i | awk -F'/' ' { printf("%s",$NF)} ')
  image_file="${image}.tar.gz"
  dest_path=$(echo $i | sed "s#docker.repository.cloudera.com#${DOCKER
_REGISTRY_DEST}#")

  docker load -i ${image_file}
  docker tag ${i} "${dest_path}"
  docker push "${dest_path}"
done
```

8. Make the newly added Runtime images available by following the instructions in Adding ML Runtimes using Runtime Repo files on page 4.

For Non-airgapped installations (with Internet access)

In non-airgapped installations (with available Internet access) upgrade ML Runtimes by setting and enabling the following options:

- Download the most recent manifest and populate the Runtime Catalog by selecting Site Administration Runtime Update Runtimes now .
- Enable automated daily upgrades to download ML Runtime images by selecting the checkbox at Site AdministrationRuntimeEnable Runtime Updates.

ML Runtimes versus Legacy Engine

While Runtimes and the Legacy Engine are both container images that contain the Linux OS, interpreter(s), and libraries, ML Runtimes keeps the images small and improves performance, maintenance, and security.

Note: Legacy Engines are deprecated since June 2021 and will be removed in a future release. Starting with version 1.5.1, on new workbenches Legacy Engines are disabled by default and no Legacy Engine image is registered in the workbench. Cloudera recommends using ML Runtimes for all new projects, and urges customers to migrate existing Engine-based projects to ML Runtimes.

Runtimes and the Legacy Engine serve the same basic goal: they are container images that contain a complete Linux OS, interpreter(s), and libraries. They are the environment in which your code runs. However, ML Runtimes design keeps the images small, which improves performance, maintenance, and security.

There is one Legacy Engine. The Engine is monolithic. It contains the machinery necessary to run sessions using all four Engine interpreter options that Cloudera currently supports (Python 2, Python 3, R, and Scala) and a much larger set of UNIX tools including LaTeX. The Conda package manager was available in the Legacy Engine. Conda is not available in ML Runtimes.

Runtimes are the future of Cloudera AI. There are many Runtimes. Currently each Runtime contains a single interpreter (for example, Python 3.8, R 4.0) and a set of UNIX tools including `gcc`. Each Runtime supports a single UI for running code (for example, the Workbench or JupyterLab).

To migrate from Legacy Engine to Runtimes, you will need to modify your project settings. See *Modifying Project Settings* for more information.

Disable Engines

Starting with version 1.5.1, Legacy Engines are disabled by default on the workbench level. This setting is accessible in Site Administration Runtime Disable Engines . Select this checkbox on upgraded workbenches to disable Legacy Engines and change all existing project types to ML Runtimes. Disabling Legacy Engines prohibits changing the project type to Legacy Engine.

This has the following effect for workloads using Legacy Engine kernels:

- Model builds cannot be redeployed
- Applications cannot be restarted
- Jobs cannot be started, scheduled jobs will fail
- Sessions cannot be started

To re-enable Legacy Engines go to Site Administration Runtime and deselect the Disable Engines checkbox under Engine Images.

Jupyter

Our Python Runtimes support JupyterLab, a general purpose IDE from the Jupyter project. The engine supports Jupyter Notebook, a simpler UI focused on Notebooks. If you prefer the simpler Notebook UI, choose Classic Notebook from the JupyterLab Help menu. To further customize the JupyterLab experience on Cloudera AI see *Using Editors for ML Runtimes*.

Build dependencies

Runtimes generally include fewer UNIX tools than the Legacy Engine. This means you are more likely to find that you cannot install a Python or R package because the Runtime is missing a build dependency such as a library. This should not happen often with Python. Most Python packages are distributed as precompiled “wheels”, so there are no build dependencies. It is more likely to happen with R packages because precompiled packages are not available for our architecture. In case of further difficulties with the build, contact customer support.

Using pip to install libraries in Python

To install a Python library from within Workbench or JupyterLab we recommend you use %pip (for example, %pip install sklearn. %pip is a command that is guaranteed to point to the right version of pip. This is optimal to be used outside of Cloudera AI as well.

Note: You do not need to add 3 to install a Python 3 library.

If you prefer to use the pip executable directly, both pip and pip3 work. This is because Runtimes do not include Python 2. Like any shell command, precede it with “!” to run it from within Workbench or JupyterLab (for example, !pip install sklearn. In the Legacy Engine you must use pip3 to install Python 3 packages and the %pip magic command is not supported.

Python paths

Python Runtimes include preinstalled Python packages at /usr/local/lib/python/<version>/site-packages. The pre-installed packages and versions are documented in *Pre-Installed Packages in ML Runtimes*.

When you use pip, you install packages into the current project (not a runtime image) at /home/cdsw/.local/lib/python/<version>/site-packages. This means you need to reinstall packages if you change Python versions.

In most cases, you can install a newer version of a package preinstalled in /usr/local into your project. For example, we preinstall numpy and you can install a newer version. But there are some exceptions to this: if you install matplotlib, ipykernel, or its dependencies (ipython, traitlets, jupyter_client, and tornado) then you may break your ability to launch sessions.

If you accidentally install these packages (or you see unexpected behavior when you switch a project from Legacy Engine to Runtimes), the simplest solution is to delete /home/cdsw/.local/lib/python and reinstall your project’s dependencies from the project overview page.

R paths

R Runtimes include preinstalled R packages at /usr/local/lib/R/library/. The pre-installed packages and versions are documented in *Pre-Installed Packages in ML Runtimes*.

When you use install.packages(), you install packages into the current project (not a runtime image) at /home/cdsw/.local/lib/R/<version>/library (for example, $R_LIBS_USER). This means you need to reinstall packages if you change R versions.

Note the R project package path in Legacy Engines. If you use engines, you install packages to /home/cdsw/R. The change to /home/cdsw/.local/lib/R/<version>/library was made to support multiple versions of R.

In most cases, you can install a newer version of a package preinstalled /usr/local into your project. For example, we preinstall ggplot2 and you can install a newer version. But there are two exceptions to this. If you install Cairo or RServe they may break your ability to launch sessions.

If you accidentally install these packages (or you see unexpected behavior when you switch a project from Legacy Engine to Runtimes), the simplest solution is to delete /home/cdsw/.local/lib/python and reinstall your project's dependencies from the project overview page.

Related Information

Creating a Project with ML Runtimes variants

Modifying Project Settings

Using Runtime Catalog

You can use the Runtime catalog to list all runtimes that are available for your deployment. A Recommended label is added to some runtime variants recommended for use by Cloudera, so the adminstrator users can easily identify the optimal selections. Administrator users can set runtime variants as default runtime variants using the checkbox.

About this task

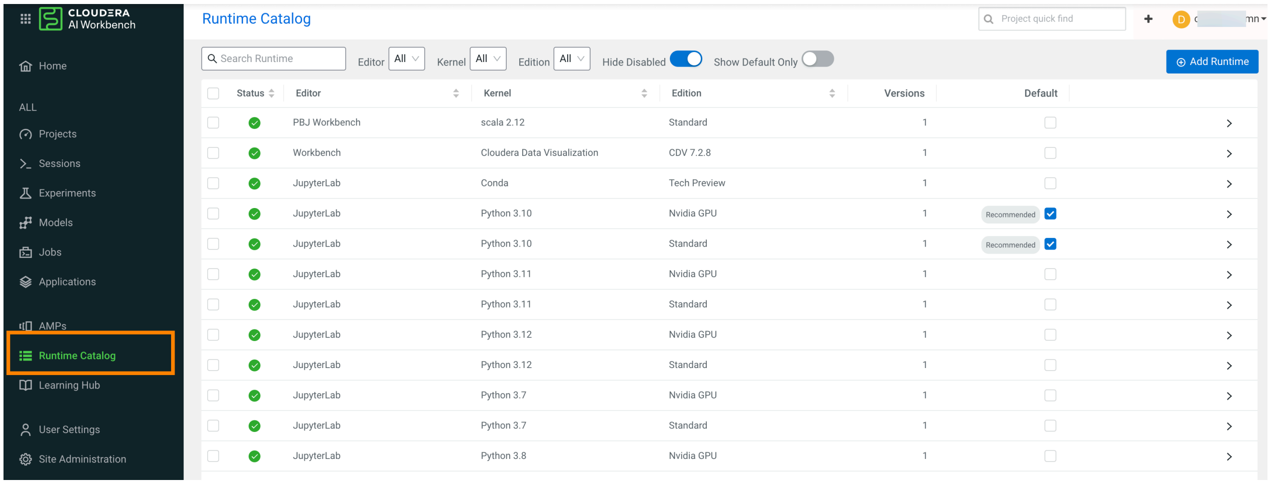
The Runtime Catalog lists information about each of the available runtimes. Information includes the editor and kernel supported by the runtime along with the edition, version, and a brief description.

Administrator users can use the Hide Disabled toggle so that only runtimes with Enabled or Deprecated status display. The toggle is on by default.

The Status filter also applies to runtime variants.

Procedure

Click Runtime Catalog in the left Navigation bar to display a list of all available runtimes.



Editor All Kernel All Edition All Hide Disabled Show Default Only

□	Status	Editor	Kernel	Edition	Versions	Default	
□	✓	PBJ Workbench	scala 2.12	Standard	1	□	>
□	✓	Workbench	Cloudera Data Visualization	CDV 7.2.8	1	□	>
□	✓	JupyterLab	Conda	Tech Preview	1	□	>
□	✓	JupyterLab	Python 3.10	Nvidia GPU	1	Recommended ☑	>
□	✓	JupyterLab	Python 3.10	Standard	1	Recommended ☑	>
□	✓	JupyterLab	Python 3.11	Nvidia GPU	1	□	>
□	✓	JupyterLab	Python 3.11	Standard	1	□	>
□	✓	JupyterLab	Python 3.12	Nvidia GPU	1	□	>
□	✓	JupyterLab	Python 3.12	Standard	1	□	>
□	✓	JupyterLab	Python 3.7	Nvidia GPU	1	□	>
□	✓	JupyterLab	Python 3.7	Standard	1	□	>
□	✓	JupyterLab	Python 3.8	Nvidia GPU	1	□	>

Changing Docker credential setting for ML Runtime

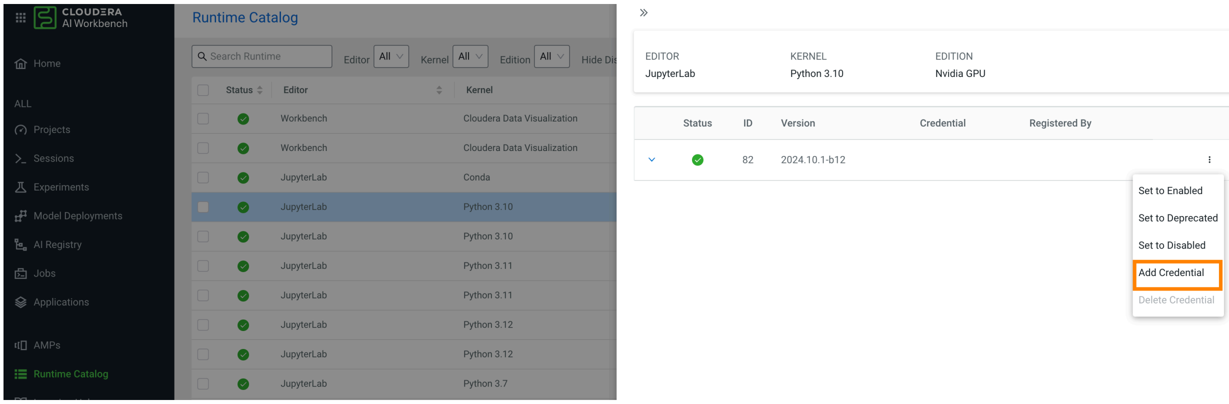
You can choose a specific Docker credential for Cloudera AI to use while fetching ML Runtime from a secure repository.

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 Runtime Catalog in the left navigation menu to display all available runtimes.
4. Select Actions > Add Credential.



Runtime Catalog

Status	Editor	Kernel
✓	Workbench	Cloudera Data Visualization
✓	Workbench	Cloudera Data Visualization
✓	JupyterLab	Conda
✓	JupyterLab	Python 3.10
✓	JupyterLab	Python 3.10
✓	JupyterLab	Python 3.11
✓	JupyterLab	Python 3.11
✓	JupyterLab	Python 3.12
✓	JupyterLab	Python 3.12
✓	JupyterLab	Python 3.7

EDITOR	KERNEL	EDITION
JupyterLab	Python 3.10	Nvidia GPU

Status	ID	Version	Credential	Registered By
✓	82	2024.10.1-b12		

5. In the **Add Credential** window, select the Docker credential you want to use.
6. Click Add.

Enabling, disabling, and deprecating Runtimes

A key feature of the Runtime Catalog is the ability to enable, disable or deprecate one or more Runtimes at once.

About this task

You can enable, disable or set one or many Runtimes deprecated at once.

Note: Runtimes cannot be deleted from the **Runtime Catalog**. Instead, you can disable a Runtime by following the instructions and then ensure that the Hide Disabled toggle is set, so that the disabled runtimes are no longer visible.

Procedure

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

   The **Home** page displays.
2. Select the required Workbench.

   The **Cloudera AI Workbench** page displays.
3. Select Runtime Catalog in the left navigation menu to display all available Runtimes.

   All available Runtimes are listed on the page.
4. Select one or more Runtimes.
5. Select the Actions button.

   The following actions are available with the Runtimes in the drop-down menu:

   - Set to Enabled
   - Set to Deprecated
   - Set to Disabled
   - Add Credential
6. Select the required action.

Using ML Runtimes addons

ML Runtime addons allow you to add Spark and Hadoop CLI to sessions run on projects using ML Runtime images.

While legacy engines include support for both Spark and Hadoop CLI, ML Runtimes do not contain Spark and Hadoop CLI binaries to keep them small and lean. Instead Spark and Hadoop binaries are stored in persistent storage on an NFS server and can be added to your ML Runtime sessions.

Adding Hadoop CLI to ML Runtime sessions

Hadoop CLI can be enabled only on sessions that are selected to use Spark.

About this task

To add Hadoop CLI to sessions run on projects using ML Runtimes images:

Procedure

1. You can view all available ML Runtime addons by selecting the Site Administration>Runtime/Engine tab and viewing Runtime Addons.
2. To include Hadoop in all sessions created in the workbench, under Site Administration>Runtime/Engine, choose the desired Hadoop version in the pull down menu next to Hadoop CLI Version.

   Note: Hadoop CLI can be enabled only on sessions that are selected to use Spark.

   This will add Hadoop CLI to all sessions created in the workbench.
3. To use Hadoop commands for your session, click the Terminal Access button at the top of the Session window.

   Cloudera AI launches a terminal window in which you can use Hadoop commands.

Adding Spark to ML Runtime Sessions

You can add Spark to ML Runtime sessions using the ML Runtimes addons. Both Spark and Hadoop CLI are enabled when you enable Spark.

About this task

Custom Spark settings can also be configured at the workbench level. When set, the custom Spark configuration provided by the administrator will be merged with the default Spark configuration used in Cloudera AI sessions. These settings will automatically apply to all newly launched Spark sessions within the workbench. These configuration settings are available under **Site Administration > Runtimes > Spark Configuration**.

To add Spark to sessions run on projects using ML Runtimes images:

Procedure

1. View all available ML Runtime addons by selecting the Site Administration > Runtimes.
2. To enable the default Spark configuration, start a New Session for an ML Runtimes project.
3. Click the Enable Spark option, then select the Spark version.
4. Click Start Session.
5. You can now run a Spark job for your session.

   The Logs tab displays the executors in your Spark job.

6. After you start a Spark job, you can access the Spark UI by clicking the Spark UI button at the top of the Session window.

Turning off ML Runtimes Addons

ML Runtimes Addons is turned on by default. However, you can disable ML Runtimes Addons.

Procedure

1. Select Site Administration > Settings.
2. Under Feature Flags, uncheck the checkbox next to Allow users to Run ML Runtimes Addons.

Customized Runtimes

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

Note: 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, Cloudera AI 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 the ML Runtime images provided by Cloudera.

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

Creating a Dockerfile for the custom Runtime Image

Follow the instructions to create a Dockerfile for a custom image.

Procedure

1. Select the source image for your customization.

   For a non-PBJ Runtime, you must use a Runtime image released by Cloudera. Image tags can be checked on the Session Start page on the user interface when you select a Runtime.
2. Create a Dockerfile when building a customized image, 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 a ML Runtime image released by Cloudera.

```
# Dockerfile
# Specify an ML Runtime base image
FROM docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-jupyterlab-
python3.7-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_MAINTEN
ANCE_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

Customizing the editor

A third-party editor can be customized to work with ML Runtimes.

Procedure

1. For a third-party editor to work with ML Runtimes, provide a starting script.

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

2. For Cloudera AI 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. The code to be run must be trusted code.

Building the new Docker Image

Follow the instructions to use Docker to build a custom image.

About this task

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.

Procedure

1. To install binaries run this command on the host where you want to build the new image.

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

2. Add the --network=host option to the build command if your Dockerfile makes any outside connection (for example, apt-get, update, pip install, curl).

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

Distributing the ML Runtime Image

Select a method 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:

- Option 1 - Push the image to a public registry such as DockerHub.

  This option is recommended for user-created runtime images without any proprietary code or settings. Typically, this is the quickest and easiest way to start using custom runtimes. For details on pushing image to Docker registry, see Docker image push.

  Note: The public registry must be available without any user or password restrictions, otherwise containerd will not be able to fetch the image.

- Option 2 - Push the image to your company's Docker registry.

  Note: This registry must be defined in Cloudera Manager, for example in Airgapped installation, or the internal registry must not be password-protected, (and its certificate must be distributed).

  You can install your own custom docker registry by following the CNCF Distribution - Distribution Registry. Also see, Creating an internal secure Docker Registry for CDSW/ML Runtimes.

  1. Tag your image with the following schema:

     ```
     docker tag [***IMAGENAME***] [***COMPANY REGISTRY***]/[***USERNAME***]/[
     ***IMAGENAME***]:[***TAG***]
     ```

  2. Once the image has been tagged properly, push the image with the following command:

     ```
     docker push [***COMPANY REGISTRY***]/[***USERNAME***]/[***IMAGENAME***]:
     [***TAG***]
     ```

  3. Add your docker registry certificate to Cloudera AI if necessary.

- Option 3 - Push the image to the Cloudera Embedded Container Service docker registry.

  This is not recommended for production environment.

  It is possible to use the Cloudera Embedded Container Service docker registry that the rest of the your Cloudera AI cluster uses for the internal and system images:

  1. Find the host where the registry is running. Run this command on any host:ls -al /etc/docker/certs.d This will contain an entry for the host running the registry:

     ```
     [root@host-1.company.com ~]$ ls -al /etc/docker/certs.d/
     total 0
     drwxr-xr-x 4 8536 8536 101 Mar  4 13:23 .
     ```

```
drwxr-xr-x 3 root root  37 Jul 28  2023 ..
drwx------ 2 8536 8536  26 Mar  4 13:23 docker-private.infra.cloudera.
com
drwxr-xr-x 2 8536 8536  40 Mar  4 13:23 host-2.company.com:5000
```

2. View the Kubernetes secret, which contains the username and password for this registry:

   ```
   [root@host-1.company.com ~]$ kubectl get secret cdp-private-installer-do
   cker-pull-secret -n  [***Cloudera AI worspace namespace***] -o jsonpath=
   "{.data.\.dockerconfigjson}" | base64 -d

   {"auths":{[***AUTH***]:{[***USERNAME***],[***PASSWORD***],[***AUTH***]
   }}}
   ```

3. Tag your custom Runtime against the registry, log into the registry, and push the image:

   ```
   [root@host-1.company.com ~] docker tag username/imagename:1 host-2.compa
   ny.com:5000/username/imagename:1

   [root@host-1.company.com ~] docker login host-2.company.com:5000
   Username: [***USERNAME***]
   Password: [***PASSWORD***]

   [root@host-1.company.com ~] docker push host-2.company.com:5000/userna
   me/imagename:1
   ```

4. Follow the documentation in Adding docker registry credentials and certificates to add both the certificate (in this example, from /etc/docker/certs.d/host-2.company.com:5000/ca.crt) and the username and password to Cloudera AI.

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.

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, the 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.

Limitations to customized ML Runtime images

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.

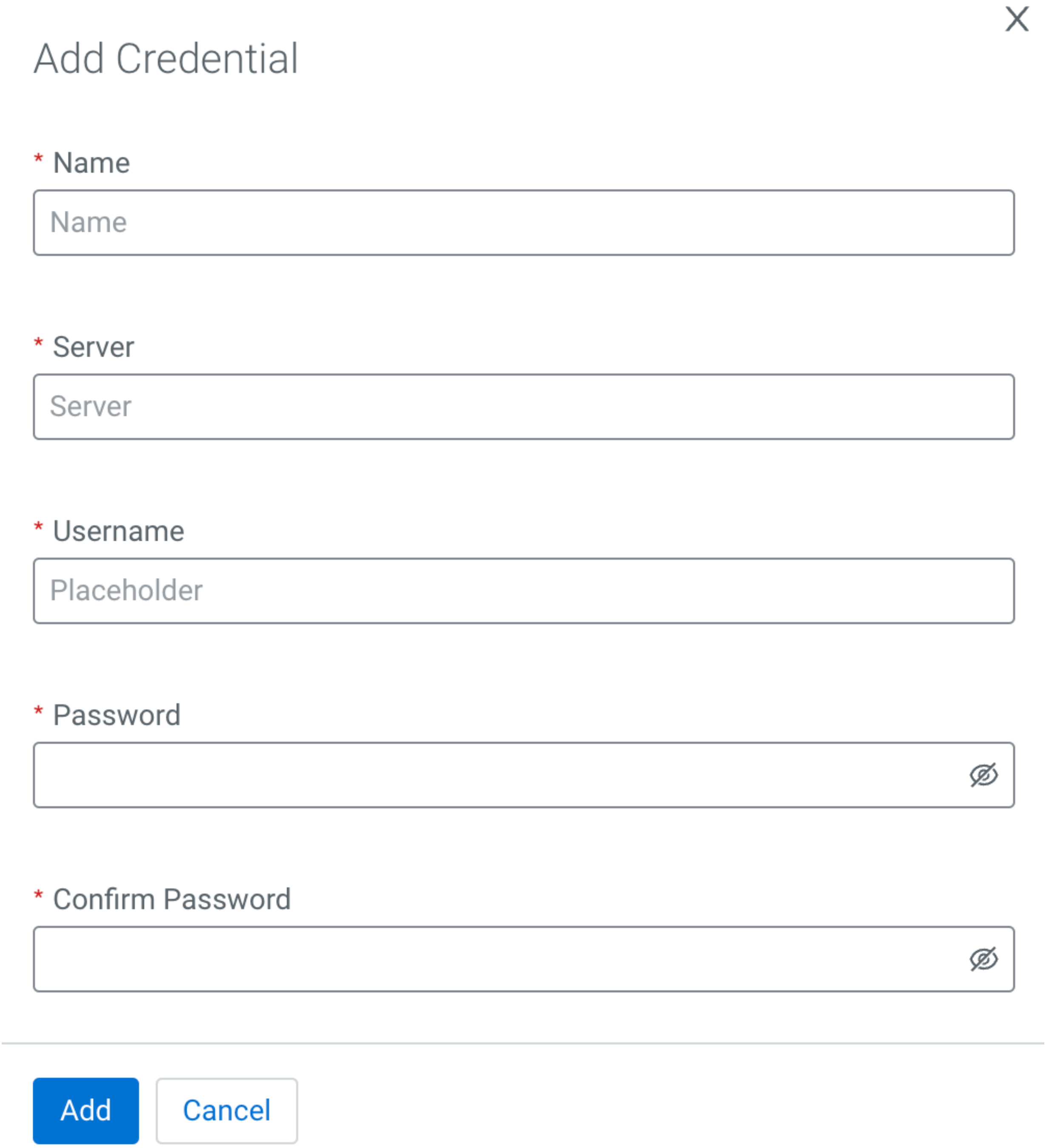
   The **Cloudera AI Workbenches** page displays.
2. Click on the name of the workbench.

   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.

For Legacy engines, the default credentials are used when Legacy Engine images are pulled. The default credential is picked up automatically for Custom Legacy Engines coming from the server stored in the default credential.

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.

Adding Docker Registry certificates to Cloudera AI

You may need to copy your registry's TLS certificates into Cloudera AI:

1. Pull the existing certificates:

```
kubectl get configmap private-cloud-ca-certs-pem -n [***CML workspace na
mespace***] -o jsonpath="{4e.binaryData.cacerts}" | base64 -d > /tmp/old
certs.pem
```

2. Append your docker registry certificates to the existing certificates and convert them to base64:

```
cat /tmp/dockercerts.crt /tmp/oldcerts.pem > /tmp/newcerts.pem
cat /tmp/newcerts.pem | base64 -w 0 > /tmp/newcerts.b64
```

3. Pull the certificates' configuration map and update it with this new certificate file:

```
kubectl get configmap private-cloud-ca-certs-pem -n [***CML workspace na
mespace***] -o yaml > /tmp/oldconfigmap.yaml

cat /tmp/oldconfigmap.yaml | sed "s/cacerts.*/cacerts: $(cat /tmp/newcerts
.b64)/" > /tmp/newconfigmap.yaml

kubectl apply -f /tmp/newconfigmap.yaml
```

4. Delete the runtime-manager pod to restart it and see these new certificates:

```
kubectl delete pod runtime-manager-xxxxxxx-xxxxx -n [***CML workspace na
mespace***]
```

Adding Docker registry certificates to Cloudera

If the Custom Runtime repository uses self-signed TLS certificates, those certificates must be trusted by Clouderaprior to creating the Cloudera AI Workbench. The certificates must be uploaded to Miscellaneous category, as uploading them to other categories might intervene into other processes. See the necessary steps to make Cloudera trust the self-signed certificates in Updating TLS certificates.

Related Information

Updating TLS certificates