

Managing Engines in Cloudera Machine Learning

Date published: 2020-07-16

Date modified: 2022-07-21



Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Basic Concepts and Terminology.....	5
ML Runtimes versus Legacy Engine.....	7
Engine Dependencies.....	8
Engines for Experiments and Models.....	10
Snapshot Code.....	11
Build Image.....	11
Run Experiment / Deploy Model.....	12
Environmental Variables.....	13
Managing Engines.....	13
Creating Resource Profiles.....	13
Configuring the Engine Environment.....	14
Set up a custom repository location.....	15
Installing Additional Packages.....	15
Using Conda to Manage Dependencies.....	16
Engine Environment Variables.....	18
Engine Environment Variables.....	19
Accessing Environmental Variables from Projects.....	20
Customized Engine Images.....	21
Creating a Customized Engine Image.....	21
Create a Dockerfile for the Custom Image.....	22
Build the New Docker Image.....	22
Distribute the Image.....	22
Including Images in allowlist for Cloudera Machine Learning projects.....	23
Add Docker registry credentials.....	23
Limitations.....	23
End-to-End Example: MeCab.....	23
Pre-Installed Packages in Engines.....	25
Base Engine 15-cml-2021.09-1.....	25
Base Engine 14-cml-2021.05-1.....	26
Base Engine 13-cml-2020.08-1.....	28
Base Engine 12-cml-2020.06-2.....	30

Base Engine 11-cml1.4.....	31
Base Engine 10-cml1.3.....	33
Base Engine 9-cml1.2.....	34

Basic Concepts and Terminology

We recommend using ML Runtimes for all new projects. You can migrate existing Engine-based projects to ML Runtimes. Engines are still supported, but new features are only available for ML Runtimes.

In the context of Cloudera Machine Learning, engines are responsible for running data science workloads and intermediating access to the underlying cluster. Cloudera Machine Learning uses Docker containers to deliver application components and run isolated user workloads. On a per project basis, users can run R, Python, and Scala workloads with different versions of libraries and system packages. CPU and memory are also isolated, ensuring reliable, scalable execution in a multi-tenant setting.

Cloudera Machine Learning engines are responsible for running R, Python, and Scala code written by users. You can think of an engine as a virtual machine, customized to have all the necessary dependencies while keeping each project's environment entirely isolated.

To enable multiple users and concurrent access, Cloudera Machine Learning transparently subdivides and schedules containers across multiple hosts. This scheduling is done using Kubernetes, a container orchestration system used internally by Cloudera Machine Learning. Neither Docker nor Kubernetes are directly exposed to end users, with users interacting with Cloudera Machine Learning through a web application.

Base Engine Image

The base engine image is a Docker image that contains all the building blocks needed to launch a Cloudera Machine Learning session and run a workload. It consists of kernels for Python, R, and Scala along with additional libraries that can be used to run common data analytics operations. When you launch a session to run a project, an engine is kicked off from a container of this image. The base image itself is built and shipped along with Cloudera Machine Learning.

Cloudera Machine Learning offers legacy engines and Machine Learning Runtimes. Both legacy engines and ML Runtimes are Docker images and contain OS, interpreters, and libraries to run user code in sessions, jobs, experiments, models, and applications. However, there are significant differences between these choices. See *ML Runtimes versus Legacy Engines* for a summary of these differences.

New versions of the base engine image are released periodically. However, existing projects are not automatically upgraded to use new engine images. Older images are retained to ensure you are able to test code compatibility with the new engine before upgrading to it manually.

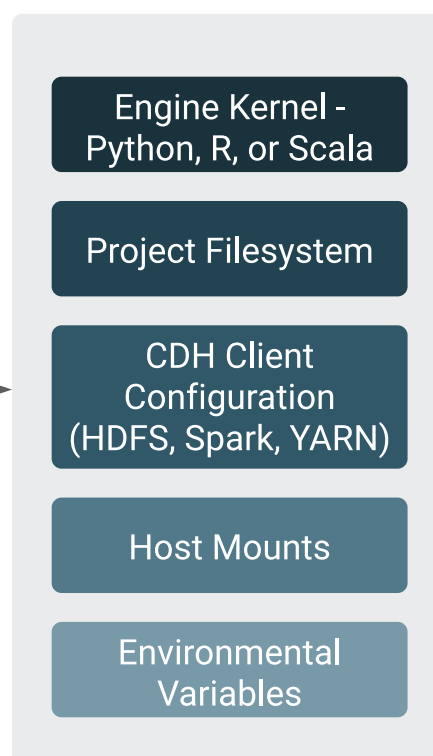
Engine

The term engine refers to a virtual machine-style environment that is created when you run a project (via session or job) in Cloudera Machine Learning. You can use an engine to run R, Python, and Scala workloads on data stored in the underlying CDH cluster.

Cloudera Machine Learning allows you to run code using either a session or a job. A session is a way to interactively launch an engine and run code while a job lets you batch process those actions and schedule them to run recursively. Each session and job launches its own engine that lives as long as the workload is running (or until it times out).

A running engine includes the following components:

Engine Environment



- Kernel

Each engine runs a kernel with an R, Python or Scala process that can be used to run code within the engine. The kernel launched differs based on the option you select (either Python 2/3, PySpark, R, or Scala) when you launch the session or configure a job.

The Python kernel is based on the Jupyter IPython kernel; the R kernel is custom-made for CML; and the Scala kernel is based on the Apache Toree kernel.

- Project Filesystem Mount

Cloudera Machine Learning uses a persistent filesystem to store project files such as user code, installed libraries, or even small data files. Project files are stored on the master host at `/var/lib/cdsw/current/projects`.

Every time you launch a new session or run a job for a project, a new engine is created, and the project filesystem is mounted into the engine's environment at `/home/cdsw`. Once the session/job ends, the only project artifacts that remain are a log of the workload you ran, and any files that were generated or modified, including libraries you might have installed. All of the installed dependencies persist through the lifetime of the project. The next time you launch a session/job for the same project, those dependencies will be mounted into the engine environment along with the rest of the project filesystem.

- Host Mounts

If there are any files on the hosts that should be mounted into the engines at launch time, use the Site Administration panel to include them.

For detailed instructions, see *Configuring the Engine Environment*.

Related Information

[ML Runtimes versus Legacy Engines](#)

[Configuring the Engine Environment](#)

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: Starting with the current CML release, Engines are deprecated. Cloudera recommends using ML Runtimes for all new projects from now on. You can also migrate existing Engine-based projects to ML Runtimes. Engines are still supported, but new features are only be available for 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.

Runtimes are the future of CML. 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'll need to modify your project settings. See *Modifying Project Settings* for more information.

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 CML 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. We have tried to cover most common use cases, but if you find you cannot build something, then please 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 “magic” command that is guaranteed to point to the right version of pip. This is a good habit to get into, as it will work outside CML. 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

ib, 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.

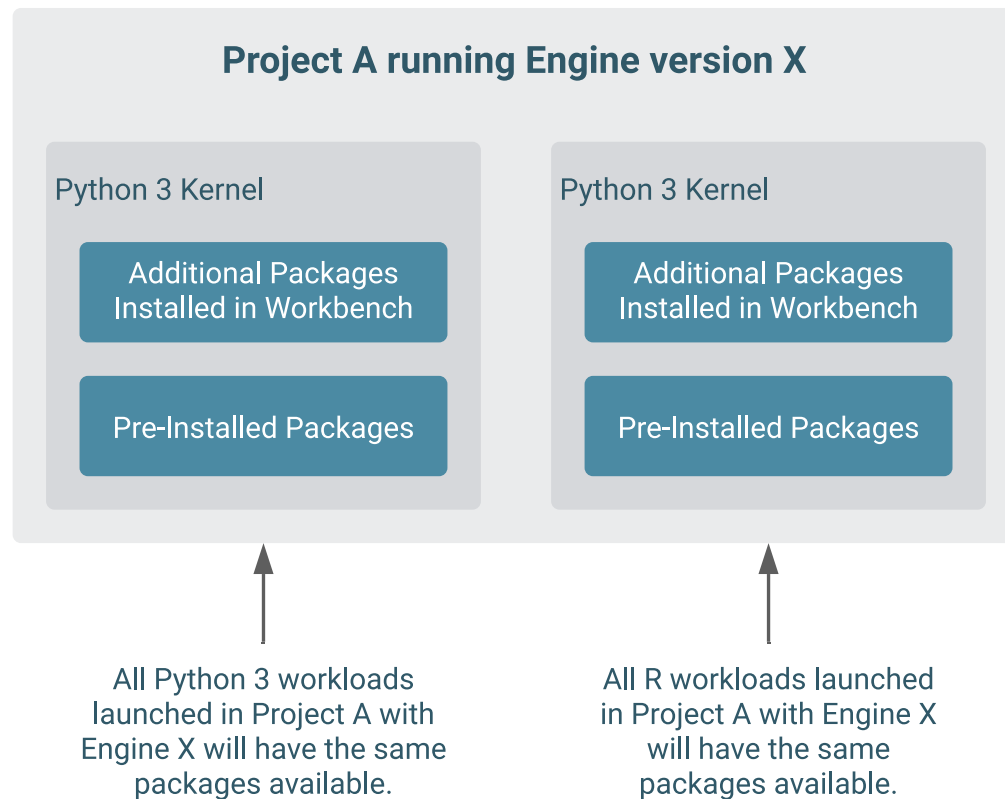
Engine Dependencies

This topic describes the options available to you for mounting a project's dependencies into its engine environment. Depending on your projects or user preferences, one or more of these methods may be more appropriate for your deployment.

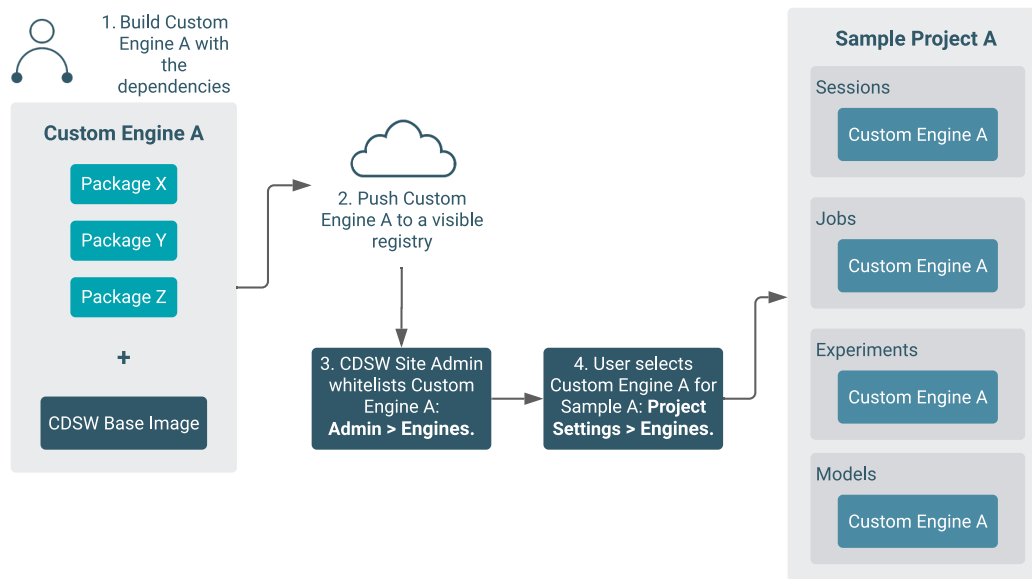


Important: Even though experiments and models are created within the scope of a project, the engines they use are completely isolated from those used by sessions or jobs launched within the same project. For details, see *Engines for Experiments and Models*.

Installing Packages Directly Within Projects



Creating a Customized Engine with the Required Package(s)



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 recommends you extend the base Cloudera Machine Learning engine image to build a customized image with all the required packages installed to it.

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 out of the box or if a package just has a complicated setup, it might be easier to simply provide users with an engine environment that has already been customized for their project(s).

For detailed instructions with an example, see *Configuring the Engine Environment*.

Managing Dependencies for Spark 2 Projects

With Spark projects, you can add external packages to Spark executors on startup. To add external dependencies to Spark jobs, specify the libraries you want added by using the appropriate configuration parameters in a `spark-defaults.conf` file.

For a list of the relevant properties and examples, see *Spark Configuration Files*.

Managing Dependencies for Experiments and Models

To allow for versioned experiments and models, Cloudera Machine Learning executes each experiment and model in a completely isolated engine. Every time a model or experiment is kicked off, Cloudera Machine Learning creates a new isolated Docker image where the model or experiment is executed. These engines are built by extending the project's designated default engine image to include the code to be executed and any dependencies as specified.

For details on how this process works and how to configure these environments, see *Engines for Experiments and Models*.

Related Information

[Engines for Experiments and Models](#)

[Installing Additional Packages](#)

[Spark Configuration Files](#)

[Configuring the Engine Environment](#)

Engines for Experiments and Models

In Cloudera Machine Learning, models, experiments, jobs, and sessions are all created and executed within the context of a project. We've described the different ways in which you can customize a project's engine environment for sessions and jobs in *Environmental Variables*. However, engines for models and experiments are completely isolated from the rest of the project.

Every time a model or experiment is kicked off, Cloudera Machine Learning creates a new isolated Docker image where the model or experiment is executed. This isolation in build and execution makes it possible for Cloudera Machine Learning to keep track of input and output artifacts for every experiment you run. In case of models, versioned builds give you a way to retain build history for models and a reliable way to rollback to an older version of a model if needed.



The following topics describe the engine build process that occurs when you kick off a model or experiment.

Related Information

[Environmental Variables](#)

Snapshot Code

When you first launch an experiment or model, Cloudera Machine Learning takes a Git snapshot of the project filesystem at that point in time. This Git server functions behind the scenes and is completely separate from any other Git version control system you might be using for the project as a whole.

However, this Git snapshot will recognize the .gitignore file defined in the project. This means if there are any artifacts (files, dependencies, etc.) larger than 50 MB stored directly in your project filesystem, make sure to add those files or folders to .gitignore so that they are not recorded as part of the snapshot. This ensures that the experiment/model environment is truly isolated and does not inherit dependencies that have been previously installed in the project workspace.

By default, each project is created with the following .gitignore file:

```
R
node_modules
*.pyc
.*
!.gitignore
```

Augment this file to include any extra dependencies you have installed in your project workspace to ensure a truly isolated workspace for each model/experiment.

Build Image

Once the code snapshot is available, Cloudera Machine Learning creates a new Docker image with a copy of the snapshot.

The new image is based off the project's designated default engine image (configured at Project Settings Engine). The image environment can be customized by using environmental variables and a build script that specifies which packages should be included in the new image.

Environmental Variables

Both models and experiments inherit environmental variables from their parent project. Furthermore, in case of models, you can specify environment variables for each model build. In case of conflicts, the variables specified per-build will override any values inherited from the project.

For more information, see *Engine Environment Variables*.

Build Script - cdsw-build.sh

As part of the Docker build process, Cloudera Machine Learning runs a build script called cdsw-build.sh file. You can use this file to customize the image environment by specifying any dependencies to be installed for the code to run successfully. One advantage to this approach is that you now have the flexibility to use different tools and libraries in each consecutive training run. Just modify the build script as per your requirements each time you need to test a new library or even different versions of a library.



Important:

- The cdsw-build.sh script does not exist by default -- it has to be created by you within each project as needed.
- The name of the file is not customizable. It must be called cdsw-build.sh.

The following sections demonstrate how to specify dependencies in Python and R projects so that they are included in the build process for models and experiments.

Python

For Python, create a requirements.txt file in your project with a list of packages that must be installed. For example:

Figure 1: requirements.txt

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

Then, create a cdsw-build.sh file in your project and include the following command to install the dependencies listed in requirements.txt.

Figure 2: cdsw-build.sh

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

Now, when cdsw-build.sh is run as part of the build process, it will install the beautifulsoup4 and seaborn packages to the new image built for the experiment/model.

R

For R, create a script called install.R with the list of packages that must be installed. For example:

Figure 3: install.R

```
install.packages(repos="https://cloud.r-project.org", c("tidyr",
"stringr"))
```

Then, create a cdsw-build.sh file in your project and include the following command to run install.R.

Figure 4: cdsw-build.sh

```
Rscript install.R
```

Now, when cdsw-build.sh is run as part of the build process, it will install the tidyr and stringr packages to the new image built for the experiment/model.

If you do not specify a build script, the build process will still run to completion, but the Docker image will not have any additional dependencies installed. At the end of the build process, the built image is then pushed to an internal Docker registry so that it can be made available to all the Cloudera Machine Learning hosts. This push is largely transparent to the end user.



Note: If you want to test your code in an interactive session before you run an experiment or deploy a model, run the cdsw-build.sh script directly in the workbench. This will allow you to test code in an engine environment that is similar to one that will eventually be built by the model/experiment build process.

Related Information

[Configuring Engine Environment Variables](#)

Run Experiment / Deploy Model

Once the Docker image has been built and pushed to the internal registry, the experiment/model can now be executed within this isolated environment.

In case of experiments, you can track live progress as the experiment executes in the experiment's Session tab.

Unlike experiments, models do not display live execution progress in a console. Behind the scenes, Cloudera Machine Learning will move on to deploying the model in a serving environment based on the computing resources and replicas you requested. Once deployed you can go to the model's Monitoring page to view statistics on the number of requests served/dropped and stderr/stdout logs for the model replicas.

Environmental Variables

This topic explains how environmental variables are propagated through an ML workspace.

Environmental variables help you customize engine environments, both globally and for individual projects/jobs. 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. Environmental variables can also be used to assign variable names to secrets, such as passwords or authentication tokens, to avoid including these directly in the code.

For a list of the environmental variables you can configure and instructions on how to configure them, see *Engine Environment Variables*.

Related Information

[Configuring Engine Environment Variables](#)

Managing Engines

This topic describes how to manage engines and configure engine environments to meet your project requirements.

Required Role: EnvironmentAdmin

Site administrators and project administrators are responsible for making sure that all projects on the deployment have access to the engines they need. Site admins can create engine profiles, determine the default engine version to be used across the deployment, and white-list any custom engines that teams require. As a site administrator, you can also customize engine environments by setting global environmental variables and configuring any files/folders that need to be mounted into project environments on run time.

By default, Cloudera Machine Learning ships a base engine image that includes kernels for Python, R, and Scala, along with some additional libraries (see *Configuring Cloudera Machine Learning Engines* for more information) that can be used to run common data analytics operations. Occasionally, new engine versions are released and shipped with Cloudera Machine Learning releases.

Engine images are available in the Site Administrator panel at Admin Engines , under the Engine Images section. As a site administrator, you can select which engine version is used by default for new projects. Furthermore, project administrators can explicitly select which engine image should be used as the default image for a project. To do so, go to the project's Overview page and click Settings on the left navigation bar.

If a user publishes a new custom Docker image, site administrators are responsible for white-listing such images for use across the deployment. For more information on creating and managing custom Docker images, see *Configuring the Engine Environment*.

Related Information

[Configuring the Engine Environment](#)

[Installing Additional Packages](#)

Creating Resource Profiles

Resource profiles define how many vCPUs and how much memory the product will reserve for a particular workload (for example, session, job, model).

About this task

As a site administrator you can create several different vCPU, GPU, and memory configurations which will be available when launching a session/job. When launching a new session, users will be able to select one of the available resource profiles depending on their project's requirements.

Procedure

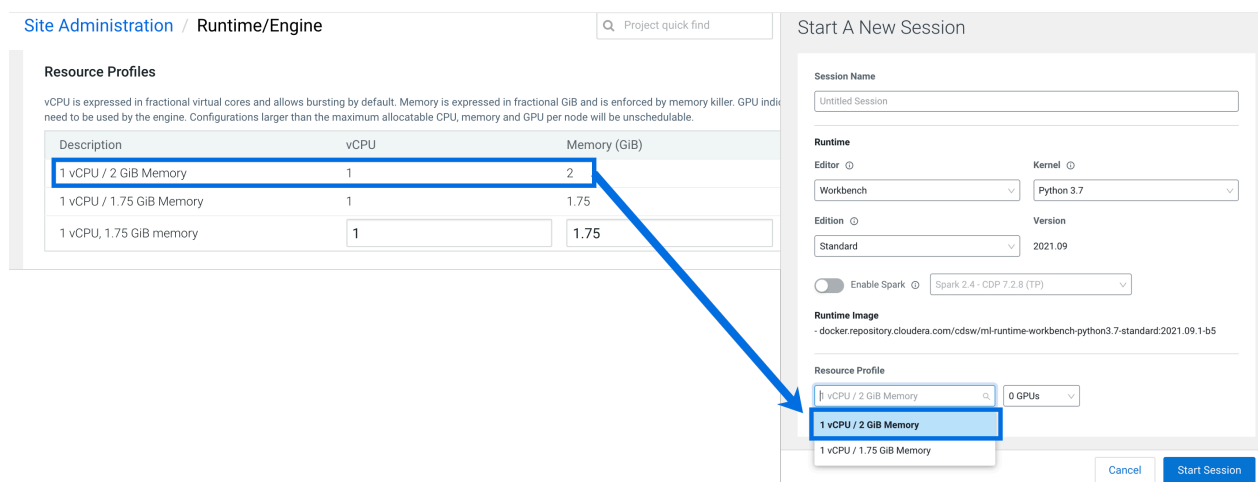
1. To create resource profiles, go to the Site Administration Runtime/Engine page.
2. Add a new profile under Resource Profiles.

Cloudera recommends that all profiles include at least 2 GB of RAM to avoid out of memory errors for common user operations.

You will see the option to add GPUs to the resource profiles only if your hosts are equipped with GPUs, and you have enabled them for use by setting the relevant properties in `cdsw.conf`.

Results

Figure 5: Resource profiles available when launching a session



Configuring the Engine Environment

This section describes some of the ways you can configure engine environments to meet the requirements of your projects.

Install Additional Packages

For information on how to install any additional required packages and dependencies to your engine, see *Installing Additional Packages*.

Environmental Variables

For information on how environmental variables can be used to configure engine environments in Cloudera Machine Learning, see *Engine Environment Variables*.

Configuring Shared Memory Limit for Docker Images

You can increase the shared memory size for the sessions, experiments, and jobs running within an Engine container within your project. For Docker, the default size of the available shared memory is 64 MB.

To increase the shared memory limit:

1. From the web UI, go to Projects Project Settings Engine Advanced Settings

2. Specify the shared memory size in the Shared Memory Limit field.
3. Click Save Advanced Settings to save the configuration and exit.

This mounts a volume with the tmpfs file system to /dev/shm and Kubernetes will enforce the given limit. The maximum size of this volume is the half of your physical RAM in the node without the swap.

Related Information

[Engine Environment Variables](#)

[Installing Additional Packages](#)

Set up a custom repository location

You can set up a custom default location for Python and R code package repositories. This is especially useful for air-gapped clusters that are isolated from the PIP and CRAN repositories on the public internet.

Python PIP repository

Custom PIP repositories can be set as default for all engines at a site or project level. The environmental variables can be set at the Project or Site level. If the values are set at the Site level, they can be overridden at the Project level.

1. Set the environmental variables at the appropriate level.
 - For Site level, go to: Site Administration Engine
 - For Project level, go to: Project Settings Engine
2. To set a new default URL for the PIP index, enter:
 - PIP_INDEX_URL = <new url>
 - PIP_EXTRA_INDEX_URL = <new url>

CRAN repository

Custom CRAN repositories must be set in a session or as part of a custom engine. To set a new default URL for a CRAN repository, set the following in the /home/cdsr/.Rprofile file:

```
options(repos=structure(c(CRAN="<mirror URL>")))
```

Installing Additional Packages

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

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 base engine image. You only need to install libraries and packages once per project. From then on, they are available to any new engine 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 Machine Learning 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 engine that extends the base engine image to include the required packages. For instructions, see *Creating a Customized Engine Image*.

(Python and R) Install Packages Using Workbench Command Prompt

To install a package from the command prompt:

1. Navigate to your project's Overview page. Click Open Workbench 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 and R have been provided.

R

```
# Install from CRAN
install.packages("ggplot2")

# Install using devtools
install.packages('devtools')
library(devtools)
install_github("hadley/ggplot2")
```

Python 2

```
# Installing from console using ! shell operator and pip:
!pip install beautifulsoup

# Installing from terminal
pip install beautifulsoup
```

Python 3

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

(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 engine, run the following command in the workbench command prompt.

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

For Python 2 engines, use pip.

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

Related Information

[Conda](#)

Using Conda to Manage Dependencies

You can install additional libraries and packages from the workbench, using either the command prompt or the terminal. Alternatively, you might choose to use a package manager such as Conda to install and maintain packages and their dependencies. This topic describes some basic usage guidelines for Conda.

Cloudera Machine Learning recommends using pip for package management along with a requirements.txt file (as described in the previous section). However, for users that prefer Conda, the default engine in Cloudera Machine Learning includes two environments called python2.7, and python3.6. These environments are added to sys.path, depending on the version of Python selected when you launch a new session.

In Python 2 and Python 3 sessions and attached terminals, Cloudera Machine Learning automatically sets the CONDA_DEFAULT_ENV and CONDA_PREFIX environment variables to point to Conda environments under /home/cdsw/.conda.

However, Cloudera Machine Learning does not automatically configure Conda to pin the actual Python version. Therefore if you are using Conda to install a package, you must specify the version of Python. For example, to use Conda to install the feather-format package into the python3.6 environment, run the following command in the Workbench command prompt:

```
!conda install -y -c conda-forge python=3.6.9 feather-format
```

To install a package into the python2.7 environment, run:

```
!conda install -y -c conda-forge python=2.7.17 feather-format
```

Note that on sys.path, pip packages have precedence over conda packages.



Note:

- Cloudera Machine Learning does not automatically configure a Conda environment for R and Scala sessions and attached terminals. If you want to use Conda to install packages from an R or Scala session or terminal, you must manually configure Conda to install packages into the desired environment.

Creating an Extensible Engine With Conda

Cloudera Machine Learning also allows you to *Configuring the Engine Environment* to include packages of your choice using Conda. To create an extended engine:

1. Add the following lines to a Dockerfile to extend the base engine, push the engine image to your Docker registry, and include the new engine in the allowlist, for your project. For more details on this step, see *Configuring the Engine Environment*.

Python 2

```
RUN mkdir -p /opt/conda/envs/python2.7
RUN conda install -y nbconvert python=2.7.17 -n python2.7
```

Python 3

```
RUN mkdir -p /opt/conda/envs/python3.6
RUN conda install -y nbconvert python=3.6.9 -n python3.6
```

2. Set the PYTHONPATH environmental variable as shown below. You can set this either globally in the site administrator dashboard, or for a specific project by going to the project's **Settings Engine** page.

Python 2

```
PYTHONPATH=$PYTHONPATH:/opt/conda/envs/python2.7/lib/python2.7/site-packages
```

Python 3

```
PYTHONPATH=$PYTHONPATH:/opt/conda/envs/python3.6/lib/python3.6/site-packages
```

Related Information

[Conda](#)

[Configuring the Engine Environment](#)

Engine Environment Variables

This topic describes how engine 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.

Environmental variables allow you to customize engine 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. Environmental variables can also be used to assign variable names to secrets such as passwords or authentication tokens to avoid including these directly in the code.

In general, Cloudera recommends that you do not include passwords, tokens, or any other secrets directly in your code because anyone with read access to your project will be able to view this information. A better place to store secrets is in your project's environment variables, where only project collaborators and admins have view access. They can therefore be used to securely store confidential information such as your AWS keys or database credentials.

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

Global

A site administrator for your Cloudera Machine Learning 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/Engines](#) .

Project

Project administrators can set project-specific environmental variables to customize the engines 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

Engines created for execution of experiments are completely isolated from the project. However, these engines 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 engines 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.

Related Information

[Basic Concepts and Terminology](#)


Engine Environment Variables

The following table lists Cloudera Machine Learning 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	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: 800,000
PROJECT_OWNER	The name of the Team or user that created the project.
SESSION_MAXIMUM_MINUTES	Maximum number of minutes a session can run before it times out. Default: 60*24*7 minutes (7 days) Maximum Value: 35,000 minutes
JOB_MAXIMUM_MINUTES	Maximum number of minutes a job can run before it times out. Default: 60*24*7 minutes (7 days) Maximum Value: 35,000 minutes
IDLE_MAXIMUM_MINUTES	Maximum number of minutes a session can remain idle before it exits. An idle session is defined as no browser interaction with the Editor. Terminal interactions are not considered as such. Contrast this to SESSION_MAXIMUM_MINUTES which is the total time the session is open, regardless of browser interaction. This variable is effective only when using the Workbench or the Jupyterlab editor. When using Cloudera's Jupyterlab Runtimes, the Editor itself is automatically configured to exit after idling for IDLE_MAXIMUM_MINUTES minutes by setting the MappingKernelManager.cull_idle_timeout and TerminalManager.cull_inactive_timeout Jupyterlab parameters accordingly. Sessions using custom Editors or the PBJ Workbench Editor do not exit due to idling. Default: 60 minutes Maximum Value: 35,000 minutes
CONDA_DEFAULT_ENV	Points to the default Conda environment so you can use Conda to install/manage packages in the Workbench. For more details on when to use this variable, see <i>Installing Additional Packages</i> .

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

Environment Variable	Description
CDSW_PROJECT	The project to which this engine belongs.
CDSW_PROJECT_ID	The ID of 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.

Environment Variable	Description
CDSW_PUBLIC_PORT	 <p>Note: This property is deprecated. See CDSW_APP_PORT and CDSW_READONLY_PORT for alternatives.</p> <p>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: <code>http(s)://read-only-<\$CDSW_ENGINE_ID>.<\$CDSW_DOMAIN></code>. By default, CDSW_PUBLIC_PORT is set to 8080.</p> <p>A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera Machine Learning web application, as long as the job or session is still running. For more details, see <i>Accessing Web User Interfaces from Cloudera Machine Learning</i>.</p> <p>In Cloudera Machine Learning, setting CDSW_PUBLIC_PORT to a non-default port number is not supported.</p>
CDSW_APP_PORT	<p>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: <code>http(s)://read-only-<\$CDSW_ENGINE_ID>.<\$CDSW_DOMAIN></code>. Use this port for applications that grant some control to the project, such as access to the session or terminal.</p> <p>A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera Machine Learning 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 <i>Accessing Web User Interfaces from Cloudera Machine Learning</i>.</p> <p>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.</p> <p>Use 127.0.0.1 as the IP.</p>
CDSW_READONLY_PORT	<p>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: <code>http(s)://read-only-<\$CDSW_ENGINE_ID>.<\$CDSW_DOMAIN></code>. Use this port for applications that grant read-only access to project results.</p> <p>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 Machine Learning 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 <i>Accessing Web User Interfaces from Cloudera Machine Learning</i>.</p> <p>Use 127.0.0.1 as the IP.</p>
CDSW_DOMAIN	The domain on which Cloudera Machine Learning is being served. This can be useful for iframing services, as demonstrated in <i>Accessing Web User Interfaces from Cloudera Machine Learning</i> .
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 Machine Learning cluster can contact this engine on this IP address.

Related Information

[Installing Additional Packages](#)

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.

R

```
database.password <- Sys.getenv( "DATABASE_PASSWORD" )
```

Python

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

Scala

```
System.getenv( "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 Engine Images

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

By default, Cloudera Machine Learning engines are preloaded with a few common packages and libraries for R, Python, and Scala. In addition to these, Cloudera Machine Learning 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 Machine Learning 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 include this new image in the allowlist for use in projects, and project administrators set the new white-listed image to be used as the default engine image for their projects. For an end-to-end example of this process, see *End-to-End Example: MeCab*.



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

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 an engine that has already been customized for their project(s).

Related Resources

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

Related Information

[End-to-End Example: MeCab](#)

[Installing Additional Packages](#)

[Customizing Docker Images in Cloudera Machine Learning](#)

Creating a Customized Engine Image

This section walks you through the steps required to create your own custom engine based on the Cloudera Machine Learning base image.

For a complete example, see *End-to-End Example: MeCab*.

Create a Dockerfile for the Custom Image

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

The first 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 beautifulsoup4 package on top of the base Ubuntu image that ships with Cloudera Machine Learning.

```
# Dockerfile

# Specify a Cloudera Machine Learning base image
FROM docker.repository.cloudera.com/cloudera/cdsw/engine:13-cml-2021.02-1

# Update packages on the base image and install beautifulsoup4
RUN apt-get update
RUN pip install beautifulsoup4 && pip3 install beautifulsoup4
```

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. 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 you want to build your image on the Cloudera Machine Learning workspace, 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 engine to all the hosts.

Once you have built a new custom engine, use one of the following ways to distribute the new image to all your Cloudera Machine Learning hosts:

Push the image to a public registry such as DockerHub

For instructions, refer 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-name>:<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>
```

The MeCab example at the end of this topic uses this method.

Related Information

[docker push](#)

Including Images in allowlist for Cloudera Machine Learning projects

This topic describes how to include custom images in the allowlist so that they can be used in projects.

Including a customized image in Cloudera Machine Learning is a two-step process.

1. Include the image in the allowlist for the whole deployment.

First, a site administrator will need to clear the new image for use on the deployment.

- a. Log in as a site administrator.
- b. Click Admin Engines .
- c. Add <company-registry>/<user-name>/<image-name>:<tag> to the allowlist of engine images.

2. Include the image in the allowlist for a specific project

If you want to start using the image in a project, the project administrator will need to set this image as the default image for the project.

- a. Go to the project Settings page.
- b. Click Engines.
- c. Select the new customized engine from the drop-down list of available Docker images. Sessions and jobs you run in your project will now have access to this engine.

Add Docker registry credentials

To enable CML to fetch custom engines from a secure repository, as Administrator you need to add Docker registry credentials.

Create a `kubectl` secret named `regcred` for your secured Docker registry. The following command creates the secret in your Kubernetes cluster:

```
kubectl create secret docker-registry regcred
  --docker-server=<server host>
  --docker-username=<username>
  --docker-password=<password>
  -n <compute namespace eg. mlx>
```

The next time the engine image is pulled, the new secret will be picked up.

Limitations

This topic lists some limitations associated with custom engines.

- Cloudera Machine Learning only supports customized engines that are based on the Cloudera Machine Learning base image.
- Cloudera Machine Learning does not support creation of custom engines larger than 10 GB.

Cloudera Bug: DSE-4420

- Cloudera Machine Learning does not support pulling images from registries that require Docker credentials.

Cloudera Bug: DSE-1521

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

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

End-to-End Example: MeCab

This topic walks you through a simple end-to-end example on how to build and use custom engines.

This section demonstrates how to customize the Cloudera Machine Learning base engine image to include the [MeCab](#) (a Japanese text tokenizer) library.

This is a sample Dockerfile that adds MeCab to the Cloudera Machine Learning base image.

```
# Dockerfile

FROM docker.repository.cloudera.com/cloudera/cds-engine:13-cml-2021.02-1
RUN rm /etc/apt/sources.list.d/*
RUN apt-get update && \
    apt-get install -y -q mecab \
        libmecab-dev \
        mecab-ipadic-utf8 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
RUN cd /tmp && \
    git clone --depth 1 https://github.com/neologd/mecab-ipadic-neologd.git \
    && \
    /tmp/mecab-ipadic-neologd/bin/install-mecab-ipadic-neologd -y -n -p /var/lib/mecab/dic/neologd && \
    rm -rf /tmp/mecab-ipadic-neologd
RUN pip install --upgrade pip
RUN pip install mecab-python==0.996
```

To use this image on your Cloudera Machine Learning project, perform the following steps.

1. Build a new image with the Dockerfile.

```
docker build --network=host -t <company-registry>/user/cds-engine-mecab:latest .
-f Dockerfile
```

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

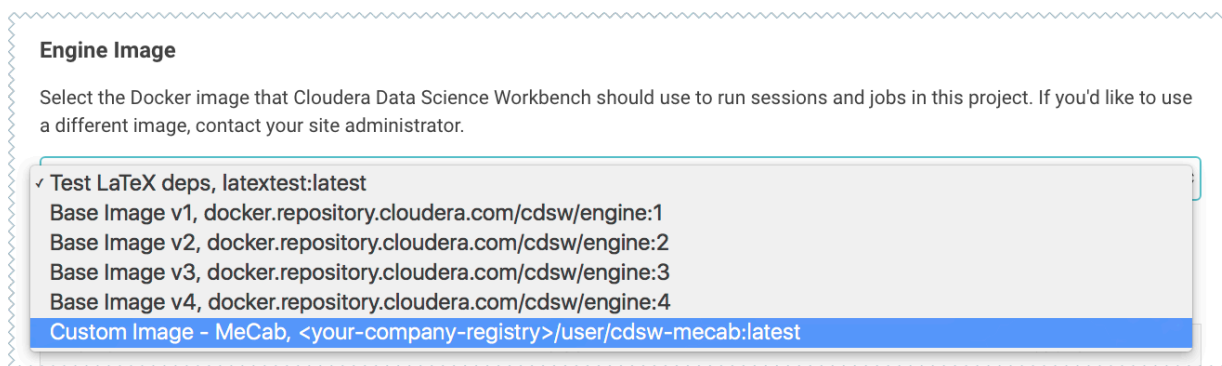
```
docker push <your-company-registry>/user/cds-engine-mecab:latest
```

3. Whitelist the image, <your-company-registry>/user/cds-engine-mecab:latest. Only a site administrator can do this.

Go to **Admin Engines** and add <company-registry>/user/cds-engine-mecab:latest to the list of whitelisted engine images.

Engine Images			
Description	Repository:Tag	Default	Actions
Test LaTeX deps	latex-test:latest	<input type="radio"/>	Edit Deprecate
Base Image v1	docker.repository.cloudera.com/cds-engine:1	<input type="radio"/>	Edit Deprecate
Base Image v2	docker.repository.cloudera.com/cds-engine:2	<input type="radio"/>	Edit Deprecate
Base Image v3	docker.repository.cloudera.com/cds-engine:3	<input type="radio"/>	Edit Deprecate
Base Image v4	docker.repository.cloudera.com/cds-engine:4	<input checked="" type="radio"/>	Edit Deprecate
Custom Image - MeCab	<input type="text" value="<your-company-registry>/user/cds-engine-mecab:latest"/>	<input type="radio"/>	Add

4. Ask a project administrator to set the new image as the default for your project. Go to the project Settings, click Engines, and select company-registry/user/cdsw-mecab:latest from the dropdown.



You should now be able to run this project on the customized MeCab engine.

Pre-Installed Packages in Engines

Cloudera Machine Learning ships with several base engine images that include Python and R kernels, and frequently used libraries.

Base Engine 15-cml-2021.09-1

Engine 15 ships Python versions 2.7.18 and 3.6.13, and R version 3.6.3.

Items in bold indicate a new version since the last release.

Table 1: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
matplotlib	2.2.4
seaborn	0.9.0
cython	0.29.13
six	1.16.0

Table 2: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0

Library	Version
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0
matplotlib	2.2.4
seaborn	0.9.0
cython	0.29.13
six	1.16.0

Table 3: R Libraries

Package	Version
RCurl	1.98.1.2
caTools	1.18.0
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.4
ggplot2	3.3.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.69
dplyr	1.0.0
httr	1.4.1
httpuv	1.5.4
jsonlite	1.6.1
magrittr	1.5
knitr	1.28
purrr	0.3.4
tm	0.7.7
proxy	0.4.24
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1
rJava	0.9.12
devtools	2.3.0

Base Engine 14-cml-2021.05-1

Engine 14 ships Python versions 2.7.18 and 3.6.10, and R version 3.6.3.

Items in bold indicate a new version since the last release.

Table 4: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.17.2
pandas	0.25.1
pandas-datareader	0.8.1
py4j	0.10.8.1
matplotlib	3.1.2
seaborn	0.9.0
cython	0.29.13
six	1.15.0

Table 5: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.10
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0
matplotlib	2.2.4
seaborn	0.9.0
cython	0.29.13
six	1.15.0

Table 6: R Libraries

Package	Version
RCurl	1.98.1.2
caTools	1.18.0
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.4
ggplot2	3.3.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.69

Package	Version
dplyr	1.0.0
httr	1.4.1
httpuv	1.5.4
jsonlite	1.6.1
magrittr	1.5
knitr	1.28
purrr	0.3.4
tm	0.7.7
proxy	0.4.24
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1
rJava	0.9.12
devtools	2.3.0

Related Information

[Base Engine 9](#)

[Base Engine 10](#)

[Base Engine 11](#)

[Base Engine 12](#)

[Base Engine 13](#)

Base Engine 13-cml-2020.08-1

Engine 13 ships Python versions 2.7.18 and 3.6.10, and R version 3.6.3.

Items in bold indicate a new version since the last release.



Note: This is the only engine available on CML Private Cloud 1.0.

Table 7: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.17.2
pandas	0.25.1
pandas-datareader	0.8.1
py4j	0.10.8.1
matplotlib	3.1.2
seaborn	0.9.0
cython	0.29.13

Library	Version
six	1.15.0

Table 8: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.10
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0
matplotlib	2.2.4
seaborn	0.9.0
cython	0.29.13
six	1.15.0

Table 9: R Libraries

Package	Version
RCurl	1.98.1.2
caTools	1.18.0
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.4
ggplot2	3.3.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.69
dplyr	1.0.0
httr	1.4.1
httpuv	1.5.4
jsonlite	1.6.1
magrittr	1.5
knitr	1.28
purrr	0.3.4
tm	0.7.7
proxy	0.4.24
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1

Package	Version
rJava	0.9.12
devtools	2.3.0

Related Information

[Base Engine 9](#)

[Base Engine 10](#)

[Base Engine 11](#)

[Base Engine 12](#)

[Base Engine 14](#)

Base Engine 12-cml-2020.06-2

Engine 12 ships Python versions 2.7.18 and 3.6.10, and R version 3.6.3.

Table 10: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.17.2
pandas	0.25.1
pandas-datareader	0.8.1
py4j	0.10.8.1
matplotlib	3.1.2
seaborn	0.9.0
Cython	0.29.13

Table 11: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0
matplotlib	2.2.4
seaborn	0.9.0
Cython	0.29.13

Table 12: R Libraries

Package	Version
RCurl	1.98.1.2
caTools	1.18.0
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.4
ggplot2	3.3.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.69
dplyr	1.0.0
httr	1.4.1
httpuv	1.5.4
jsonlite	1.6.1
magrittr	1.5
knitr	1.28
purrr	0.3.4
tm	0.7.7
proxy	0.4.24
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1
rJava	0.9.12
devtools	2.3.0

Related Information[Base Engine 9](#)[Base Engine 10](#)[Base Engine 11](#)[Base Engine 13](#)[Base Engine 14](#)**Base Engine 11-cml1.4**

Engine 11 ships Python versions 2.7.17 and 3.6.9, and R version 3.6.2.

Table 13: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0

Library	Version
numpy	1.17.2
pandas	0.25.1
pandas-datareader	0.8.1
py4j	0.10.8.1
matplotlib	2.0.0
seaborn	0.9.0
Cython	0.29.13

Table 14: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0
matplotlib	2.0.0
seaborn	0.9.0
Cython	0.29.13

Table 15: R Libraries

Package	Version
RCurl	1.95.4.12
caTools	1.17.1.3
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.3
ggplot2	3.2.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.73
dplyr	0.8.3
httr	1.4.1
httpuv	1.5.2
jsonlite	1.6
magrittr	1.5
knitr	1.26
purrr	0.3.3

Package	Version
tm	0.7.7
proxy	0.4.23
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1
rJava	0.9.11
devtools	2.2.1

Related Information

[Base Engine 9](#)

[Base Engine 10](#)

[Base Engine 12](#)

[Base Engine 13](#)

[Base Engine 14](#)

Base Engine 10-cml1.3

Engine 10 ships Python versions 2.7.17 and 3.6.9, and R version 3.5.1.

Table 16: Python 3 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.17.2
pandas	0.25.1
pandas-datareader	0.8.1
py4j	0.10.8.1
matplotlib	2.0.0
seaborn	0.9.0
Cython	0.29.13

Table 17: Python 2 Libraries

Library	Version
ipython	5.1.0
requests	2.22.0
simplejson	3.16.0
numpy	1.16.5
pandas	0.24.2
pandas-datareader	0.8.0
py4j	0.10.8.1
futures	3.3.0

Library	Version
matplotlib	2.0.0
seaborn	0.9.0
Cython	0.29.13

Table 18: R Libraries

Package	Version
RCurl	1.95.4.12
caTools	1.17.1.3
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.3
ggplot2	3.2.1
cluster	2.1.0
codetools	0.2.16
foreign	0.8.73
dplyr	0.8.3
httr	1.4.1
httpuv	1.5.2
jsonlite	1.6
magrittr	1.5
knitr	1.26
purrr	0.3.3
tm	0.7.7
proxy	0.4.23
data.table	1.12.8
stringr	1.4.0
Rook	1.1.1
rJava	0.9.11
devtools	2.2.1

Related Information[Base Engine 9](#)[Base Engine 11](#)[Base Engine 12](#)[Base Engine 13](#)[Base Engine 14](#)**Base Engine 9-cml1.2**

Engine 9 ships Python 2.7.11 and 3.6.8, and R version 3.5.1.

Table 19: Python Libraries

Library	Version
ipython	5.1.0
requests	2.13.0
Flask	0.12.0
simplejson	3.10.0
numpy	1.13.3
pandas	0.20.1
pandas-datareader	0.2.1
py4j	0.10.7
futures	2.1.4
matplotlib	2.0.0
seaborn	0.8.0
Cython	0.25.2
kudu-python	1.2.0

Table 20: R Libraries

Package	Version
RCurl	1.95.4.12
caTools	1.17.1.2
svTools	0.9.5
png	0.1.7
RJSONIO	1.3.1.2
ggplot2	3.1.1
cluster	2.0.9
codetools	0.2.16
foreign	0.8.71
dplyr	0.8.1
httr	1.4.0
httpuv	1.5.1
jsonlite	1.6
magrittr	1.5
knitr	1.23
purrr	0.3.2
tm	0.7.6
proxy	0.4.23
data.table	1.12.2
stringr	1.4.0
Rook	1.1.1
rJava	0.9.11

Package	Version
devtools	2.0.2

Related Information[Base Engine 10](#)[Base Engine 11](#)[Base Engine 12](#)[Base Engine 13](#)[Base Engine 14](#)