Machine Learning 1.5.0

Cloudera Machine Learning Engines

Date published: 2020-07-16
Date modified: 2023-01-31

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2025. 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.....4

ML Runtimes versus Legacy Engine.....6

Engine Dependencies..... 7

Engines for Experiments and Models..... 9

Snapshot Code..... 10
Build Image.....10
Run Experiment / Deploy Model..... 11

Environmental Variables..... 12

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.

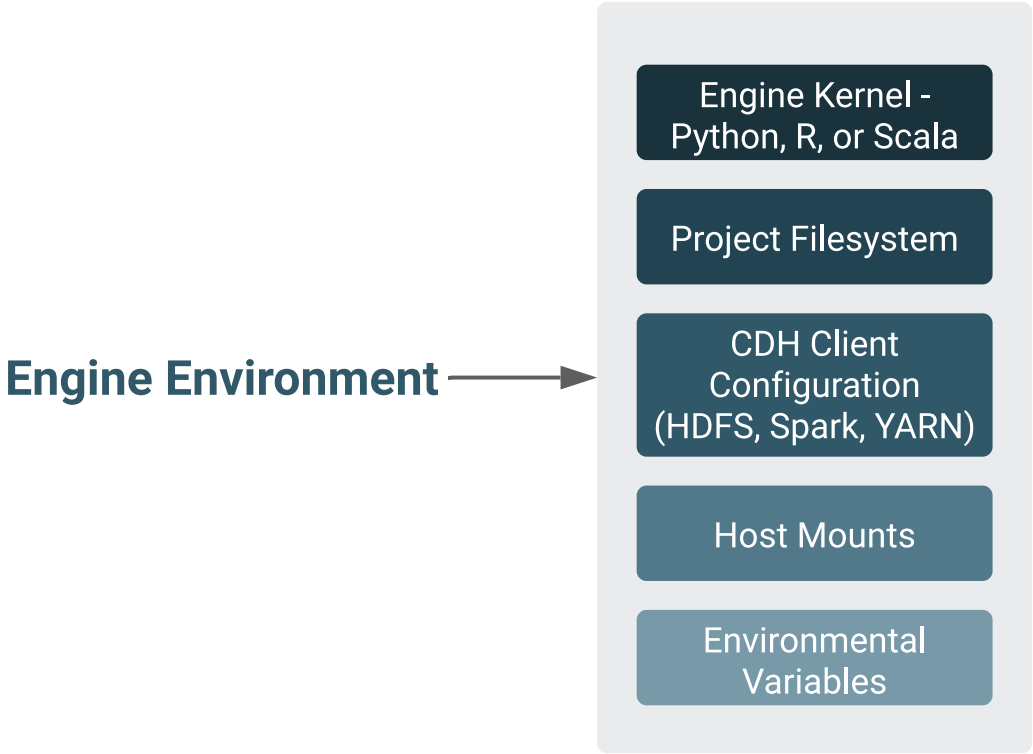
New versions of the base engine image are released periodically. However, existing projects are not automatically upgraded to use new engine images. Older images are retained to ensure you are able to test code compatibility with the new engine before upgrading to it manually.

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:



- 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. The Conda package manager was available in the Legacy Engine. Conda is not available in ML Runtimes.

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 matplotl

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/cd sw/.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.

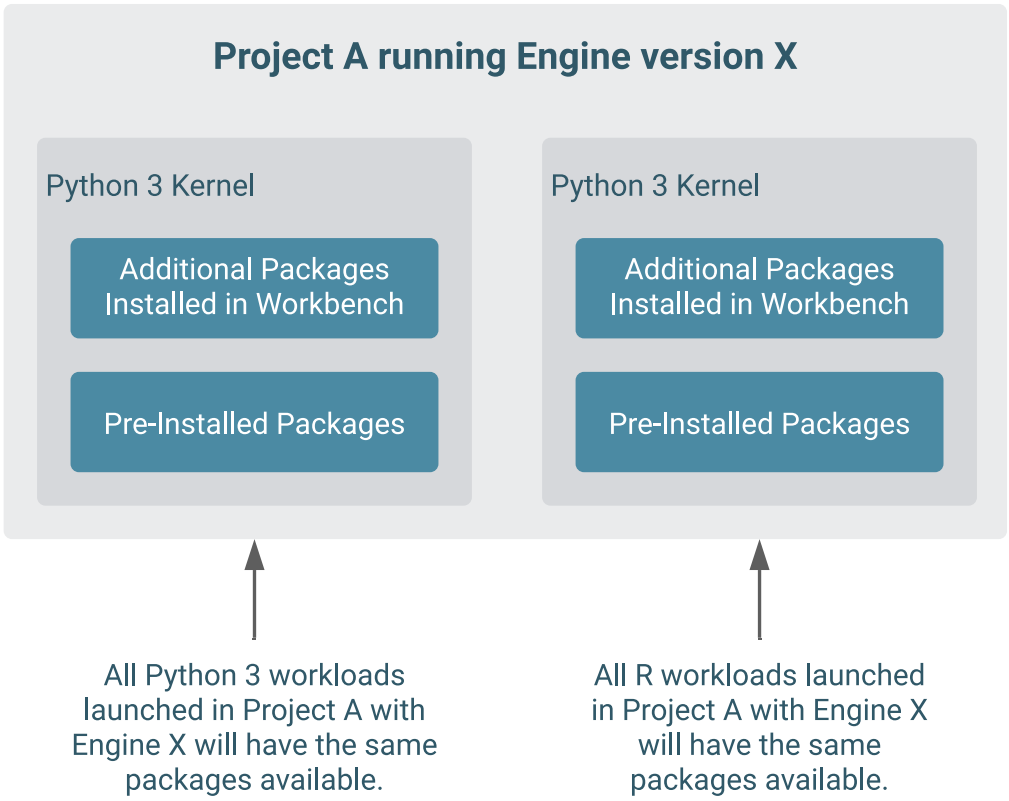
If you accidentally install these packages (or you see unexpected behavior when you switch a project from Legacy Engine to Runtimes), the simplest solution is to delete /home/cdsw/.local/lib/python and reinstall your project's dependencies from the project overview page.

Engine Dependencies

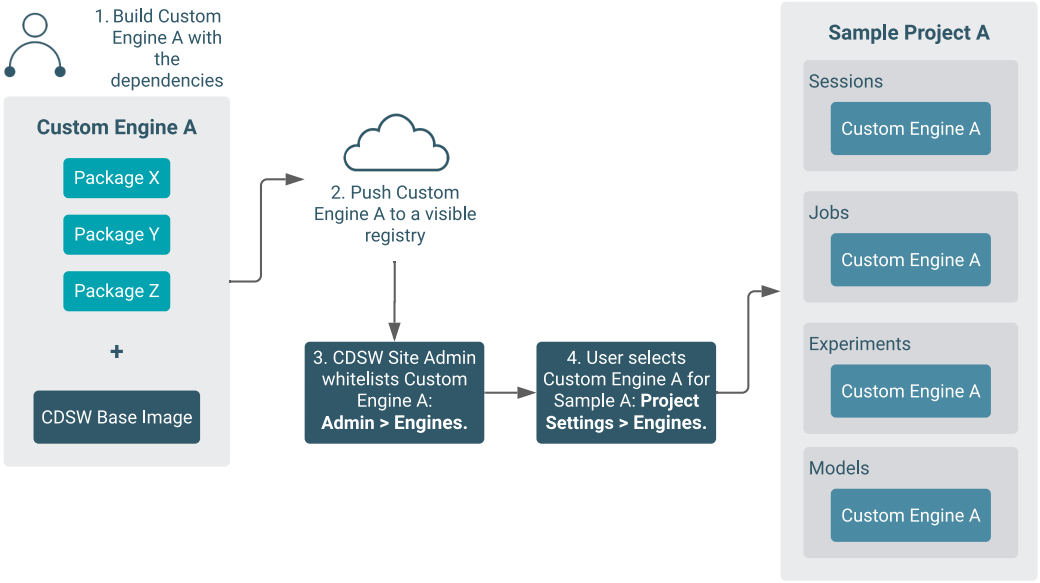
This topic describes the options available to you for mounting a project's dependencies into its engine environment. Depending on your projects or user preferences, one or more of these methods may be more appropriate for your deployment.

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*.

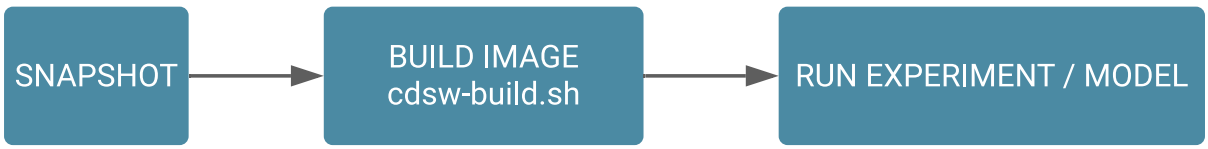
Related Information

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 inst all.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