1.3.3

Cloudera Machine Learning Private Cloud 1.3.3 User Guide

Date published: 2021-04-20
Date modified: 2021-12-17

CLOUDERA

https://docs.cloudera.com/

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

Concepts 9
Cloudera Machine Learning overview 9
AI applications 9
Exploratory Data Science 9
ML Ops 9
Core capabilities 10
Cloudera Machine Learning benefits 10
Key differences between Cloudera Machine Learning and Cloudera Data Science Workbench 11
Basic Concepts and Terminology 11
ML Runtimes versus Legacy Engine 13
Engine Dependencies 14
Engines for Experiments and Models 16
Snapshot Code 17
Build Image 17
Run Experiment / Deploy Model 18
Environmental Variables 19
Model Training and Deployment Overview 19
Experiments 19
Experiments - Concepts and Terminology 20
Models 21
Models - Concepts and Terminology 22
Collaborating on Projects with Cloudera Machine Learning 24
Project Collaborators 24
Teams 24
ML Business User 25
Forking Projects 25
Collaborating with Git 25
Sharing Job and Session Console Outputs 25
Autoscaling Workloads with Kubernetes 26

Planning 26
Introduction to Private Cloud 26
Cloudera Machine Learning requirements (OCP) 26
Cloudera Machine Learning requirements (ECS) 28
Get started with CML on Private Cloud 28
Test your connectivity to the CDP-DC cluster 29
Differences Between Public and Private Cloud 30
Limitations on Private Cloud 30
Network File System (NFS) 30
NFS Options for Private Cloud 31
Internal Network File System on OCP (Non-Production) 31
Internal Network File System on ECS (Non-Production) 32
Using an External NFS Server 33
Deploy an ML Workspace with Support for TLS 34
Replace a Certificate 34
Deploy an ML Workspace with Support for TLS on ECS 35
GPU node setup 35

How To.....36
Provision an ML Workspace.....36
Monitoring ML Workspaces.....37
Removing ML Workspaces.....37
User Roles.....38
Business Users and CML.....38
Managing your Personal Account.....39
Creating a Team.....39
Managing a Team Account.....40
Collaborating on Projects with Cloudera Machine Learning.....40
Project Collaborators.....40
Teams.....41
ML Business User.....41
Forking Projects.....41
Collaborating with Git.....41
Sharing Job and Session Console Outputs.....41
Projects in Cloudera Machine Learning.....42
Creating a Project with Legacy Engine Variants.....43
Configuring Project-level Runtimes.....44
Adding Project Collaborators.....44
Modifying Project Settings.....45
Managing Project Files.....46
Custom Template Projects.....47
Deleting a Project.....47
Native Workbench Console and Editor.....48
Launch a Session.....48
Run Code.....49
Access the Terminal.....50
Stop a Session.....50
Workbench editor file types.....51
Third-Party Editors.....51
Modes of Configuring Third-Party Editors.....52
Configure a Browser IDE as an Editor.....53
Configure a Local IDE using an SSH Gateway.....57
Configure PyCharm as a Local IDE.....58
Configure VS Code as a Local IDE.....60
Git for Collaboration.....81
Linking an Existing Project to a Git Remote.....82
Web Applications Embedded in Sessions.....82
Example: A Shiny Application.....83
Basic Concepts and Terminology.....84
ML Runtimes versus Legacy Engine.....86
Engine Dependencies.....87
Engines for Experiments and Models.....89
Snapshot Code.....90
Build Image.....90
Run Experiment / Deploy Model.....91
Environmental Variables.....92
Managing Engines.....92
Creating Resource Profiles.....92
Configuring the Engine Environment.....93
Set up a custom repository location.....94
Installing Additional Packages.....94
Using Conda to Manage Dependencies.....95

Engine Environment Variables....96
Engine Environment Variables....97
Accessing Environmental Variables from Projects....99
Customized Engine Images....100
Creating a Customized Engine Image....100
Limitations....102
End-to-End Example: MeCab....102
Pre-Installed Packages in Engines....103
Base Engine 15-cml-2021.09-1....103
Base Engine 14-cml-2021.05-1....105
Base Engine 13-cml-2020.08-1....106
Base Engine 12-cml-2020.06-2....108
Base Engine 11-cml1.4....110
Base Engine 10-cml1.3....111
Base Engine 9-cml1.2....113
Apache Spark 2 and Spark 3 on CML....114
Apache Spark supported versions....116
Spark Configuration Files....116
Managing Memory Available for Spark Drivers....116
Managing Dependencies for Spark 2 Jobs....116
Spark Log4j Configuration....117
Setting Up an HTTP Proxy for Spark 2....118
Spark Web UIs....118
Using Spark 2 from Python....118
Example: Montecarlo Estimation....119
Example: Locating and Adding JARs to Spark 2 Configuration....119
Using Spark 2 from R....120
Using Spark 2 from Scala....121
Managing Dependencies for Spark 2 and Scala....122
Using GPUs for Cloudera Machine Learning projects....122
Using GPUs with Legacy Engines....123
Custom CUDA-capable Engine Image....123
Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment....124
Project Admins: Enable the CUDA Engine for your Project....125
Testing GPU Setup....125
Running an Experiment (QuickStart)....126
Limitations....129
Tracking Metrics....129
Saving Files....130
Debugging Issues with Experiments....130
Model Training and Deployment Overview....131
Experiments....132
Experiments - Concepts and Terminology....132
Models....134
Models - Concepts and Terminology....134
Challenges with Machine Learning in production....136
Challenges with model deployment and serving....136
Challenges with model monitoring....136
Challenges with model governance....138
Creating and Deploying a Model....140
Usage Guidelines....143
Known Issues and Limitations....144
Model Request and Response Formats....145
Testing Calls to a Model....146
Securing Models....148
Access Keys for Models....148

API Key for Models....148
Workflows for Active Models....150
Technical Metrics for Models....152
Debugging Issues with Models....152
Deleting a Model....153
Example - Model Training and Deployment (Iris)....153
Train the Model....154
Deploy the Model....156
Enabling model governance....159
ML Governance Requirements....159
Registering training data lineage using a linking file....160
Viewing lineage for a model deployment in Atlas....161
Enabling model metrics....161
Tracking model metrics without deploying a model....161
Tracking metrics for deployed models....162
Analytical Applications....162
Securing Applications....164
Limitations with Analytical Applications....164
Monitoring applications....165
Creating a Job....165
Creating a Pipeline....167
Viewing Job History....167
Legacy Jobs API (Deprecated)....168
Distributed Computing with Workers....171
Workers API....171
Worker Network Communication....172
Applied ML Prototypes (AMPs)....174
Creating New AMPs....175
Custom AMP Catalog....175
Add a catalog....175
Catalog File Specification....176
AMP Project Specification....177
Host names required by AMPs....192
Managing Users....192
Configuring Quotas....193
Creating Resource Profiles....194
Onboarding Business Users....195
Adding a Collaborator....195
Monitoring Cloudera Machine Learning Activity....195
Tracked User Events....196
Monitoring User Events....199
Monitoring Active Models Across the Workspace....201
Monitoring and Alerts....202
Application Polling Endpoint....202
Choosing Default Engine....202
Controlling User Access to Features....203
Cloudera Machine Learning Email Notifications....204
Web session timeouts....205
Project Garbage Collection....205
How to make base cluster configuration changes....205
Disable Addons....206
Configuring External Authentication with LDAP and SAML....206
Configuring LDAP/Active Directory Authentication....206
Configuring SAML Authentication....208
Configuring HTTP Headers for Cloudera Machine Learning....210
Enable HTTP Security Headers....211

Enable HTTP Strict Transport Security (HSTS)....211
Enable Cross-Origin Resource Sharing (CORS)....212
SSH Keys....212
Personal Key....212
Team Key....212
Adding an SSH Key to GitHub....212
Creating an SSH Tunnel....213
Autoscaling Workloads with Kubernetes....213
Restricting User-Controlled Kubernetes Pods....213
Hadoop Authentication for ML Workspaces....214
CML and outbound network access....214

Troubleshooting....215
Troubleshooting....215
Downloading diagnostic bundles for a workspace....215
Troubleshooting Issues with Workloads....215
Troubleshooting Kerberos Errors....216

Reference....217
CML API v2....217
API v2 Usage....219
Command Line Tools in CML....222
cdswctl Command Line Interface Client....223
Download and Configure cdswctl....223
Initialize an SSH Endpoint....224
Log into cdswctl....225
Prepare to manage models using the model CLI....226
Create a model using the CLI....226
Build and deployment commands for models....228
Deploy a new model with updated resources....229
View replica logs for a model....229
cdswctl command reference....230
Data Access....230
Upload and work with local files....231
Connect to CDW....231
Accessing data with Spark....232
Connect to a CDP Data Hub cluster....234
Connect to external Amazon S3 buckets....234
Connect to External SQL Databases....235
Accessing Ozone from Spark....236
Built-in CML Visualizations....236
Simple Plots....236
Saved Images....237
HTML Visualizations....237
IFrame Visualizations....238
Grid Displays....239
Documenting Your Analysis....239
Cloudera Data Visualization for ML....240
Jupyter Magic Commands....240
Python....240
Scala....241

Release Notes....241

What's New 241
Known Issues and Limitations 244

Concepts

The Cloudera Machine Learning Data Service encompasses a wide range of functions and features for enterprise-grade data scienc and machine learning applications. The Concepts section provides overviews of the main functional areas.

Cloudera Machine Learning overview

Machine learning has become one of the most critical capabilities for modern businesses to grow and stay competitive today. From automating internal processes to optimizing the design, creation, and marketing processes behind virtually every product consumed, ML models have permeated almost every aspect of our work and personal lives.

ML development is iterative and complex, made even harder because most ML tools aren’t built for the entire machine learning lifecycle. Cloudera Machine Learning on Cloudera Data Platform accelerates time-to-value by enabling data scientists to collaborate in a single unified platform that is all inclusive for powering any AI use case. Purpose-built for agile experimentation and production ML workflows, Cloudera Machine Learning manages everything from data preparation to MLOps, to predictive reporting. Solve mission critical ML challenges along the entire lifecycle with greater speed and agility to discover opportunities which can mean the difference for your business.

Each ML workspace enables teams of data scientists to develop, test, train, and ultimately deploy machine learning models for building predictive applications all on the data under management within the enterprise data cloud. ML workspaces support fully-containerized execution of Python, R, Scala, and Spark workloads through flexible and extensible engines.

AI applications

Analytical Applications provide a place to host long running applications within a CML project.

While CML offers a place for Data Scientists to perform advanced analytics and models into production, Analytical Applications provides a place to host long running applications within a CML project. This opens up a larger group of users to the insights discovered in CML. Applications can be built with a variety of frameworks like Flask and Streamlit. They run within their own isolated compute instance which keeps them from timing out and they take advantage of ML Runtimes. Applications are accessible to users through the web. Applications can be for a variety of use cases like hosting interactive data visualizations or providing a UI frontend for a deployed mode in CML.

Exploratory Data Science

CML enables data practitioners to discover, query, and easily visualize their data sets all within a single user interface.

The beginning of every data science project begins with finding and understanding the data you need. CML brings all the tools you need for exploratory data analysis together in a single UI so that data practitioners don't have to jump between applications, and IT doesn't have to support multiple tools. CML provides users with a view of available data assets that they can connect to, a sql editor to query those data sources, and an easy-to-use drag-and-drop visualization tool to understand your data and communicate insights.

ML Ops

CML enables users to deploy machine learning and other models into production.

CML enables users to deploy machine learning and other models into production, either as a batch process through the Jobs functionality, or as near-real-time REST APIs using the Models functionality. In addition, CML provides a number of features to help maintain, monitor and govern these models in production. The Model Governance feature ensures that every deployed Model is tracked in the Cloudera Data Catalog, and allows the user to specify which data tables were used to train the model in order to provide model-data lineage. Deployed Models have a built-in dashboard for monitoring technical metrics relating to deployed CML Models, such as request throughput, latency,

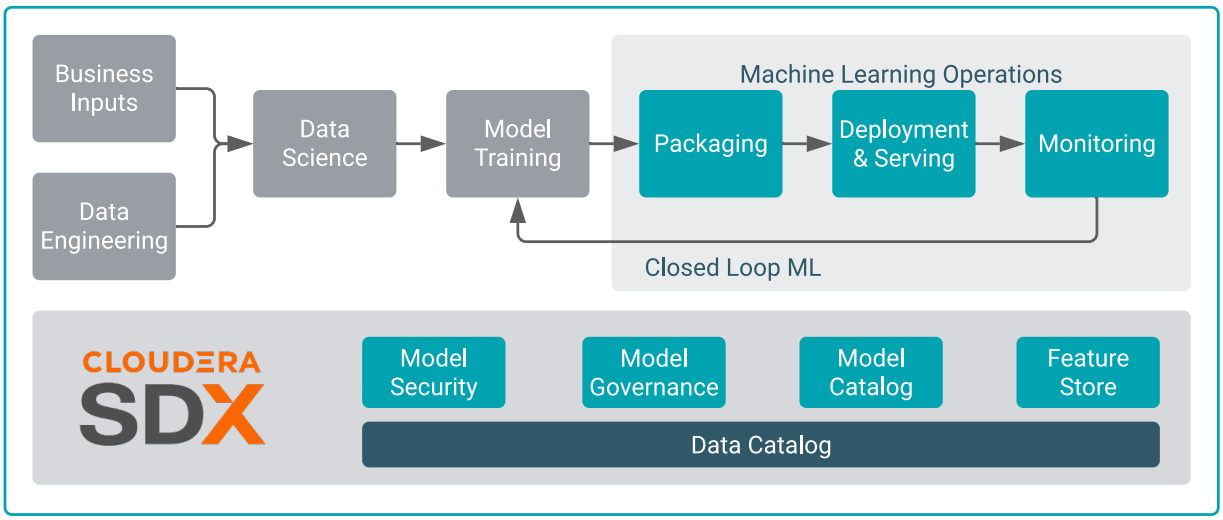
and resource consumption. Additionally, users can track arbitrary business metrics relating to each inference event, and match the results with delayed metrics from a data warehouse or other source using an automatically generated UUID. By analyzing these metrics, the user can assess the model for aggregated metrics such as accuracy on an ongoing basis.

Core capabilities

This section details the core capabilities for Cloudera Machine Learning.

Cloudera Machine Learning covers the end-to-end machine learning workflow, enabling fully isolated and containerized workloads - including Python, R, and Spark-on-Kubernetes - for scale-out data engineering and machine learning with seamless distributed dependency management.

- Sessions enable Data Scientists to directly leverage the CPU, memory, and GPU compute available across the workspace, while also being directly connected to the data in the data lake.
- Experiments enable Data Scientists to run multiple variations of model training workloads, tracking the results of each Experiment in order to train the best possible Model.
- Models can be deployed in a matter of clicks, removing any roadblocks to production. They are served as REST endpoints in a high availability manner, with automated lineage building and metric tracking for MLOps purposes.
- Jobs can be used to orchestrate an entire end-to-end automated pipeline, including monitoring for model drift and automatically kicking off model re-training and re-deployment as needed.
- Applications deliver interactive experiences for business users in a matter of clicks. Frameworks such as Flask and Shiny can be used in development of these Applications, while Cloudera Data Visualization is also available as a point-and-click interface for building these experiences.



Cloudera Machine Learning benefits

This section details the Cloudera Machine Learning benefits for each type of user.

Cloudera Machine Learning is built for the agility and power of cloud computing, but is not limited to any one provider or data source. It is a comprehensive platform to collaboratively build and deploy machine learning capabilities at scale.

Cloudera Machine Learning provides benefits for each type of user.

Data Scientists

- Enable DS teams to collaborate and speed model development and delivery with transparent, secure, and governed workflows
- Expand AI use cases with automated ML pipelines and an integrated and complete production ML toolkit

- Empower faster decision making and trust with end-to-end visibility and auditability of data, processes, models, and dashboards

IT

- Increase DS productivity with visibility, security, and governance of the complete ML lifecycle
- Eliminate silos, blindspots, and the need to move/duplicate data with a fully integrated platform across the data lifecycle.
- Accelerate AI with self-service access and containerized ML workspaces that remove the heavy lifting and get models to production faster

Business Users

- Access interactive Applications built and deployed by DS teams.
- Be empowered with predictive insights to more intelligently make business decisions.

Key differences between Cloudera Machine Learning and Cloudera Data Science Workbench

This topic highlights some key differences between Cloudera Data Science Workbench and its cloud-native counterpart, Cloudera Machine Learning.

How is Cloudera Machine Learning (CML) related to Cloudera Data Science Workbench (CDSW)?

CML expands the end-to-end workflow of Cloudera Data Science Workbench (CDSW) with cloud-native benefits like rapid provisioning, elastic autoscaling, distributed dependency isolation, and distributed GPU training.

It can run its own native distributed computing workloads without requiring a separate CDH cluster for scale-out compute. It is designed to run on CDP in existing Kubernetes environments, reducing operational costs for some customers while delivering multi-cloud portability. On Public Cloud, managed cloud Kubernetes services include EKS, AKS, or GKE, and on Private Cloud, they include Red Hat OpenShift or ECS (Embedded Container Service).

Both products help data engineers and data science teams be more productive on shared data and compute, with strong security and governance. They share extensive code.

There is one primary difference:

- CDSW extends an existing CDH cluster, by running on gateway nodes and pushing distributed compute workloads to the cluster. CDSW requires and supports a single CDH cluster for its distributed compute, including Apache Spark.
- In contrast, CML is self-contained and manages its own distributed compute, natively running workloads - including but not limited to Apache Spark - in containers on Kubernetes.

  Note: It can still connect to an existing cluster to leverage its distributed compute, data, or metadata (SDX).

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.

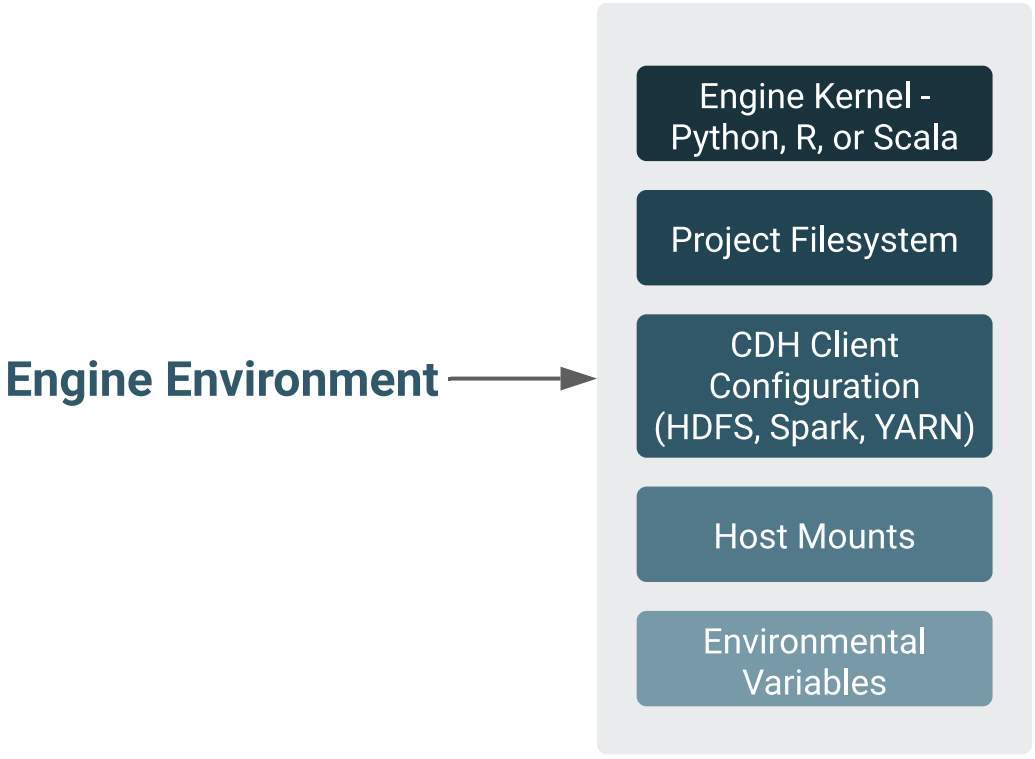
New versions of the base engine image are released periodically. However, existing projects are not automatically upgraded to use new engine images. Older images are retained to ensure you are able to test code compatibility with the new engine before upgrading to it manually.

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.

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, 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.

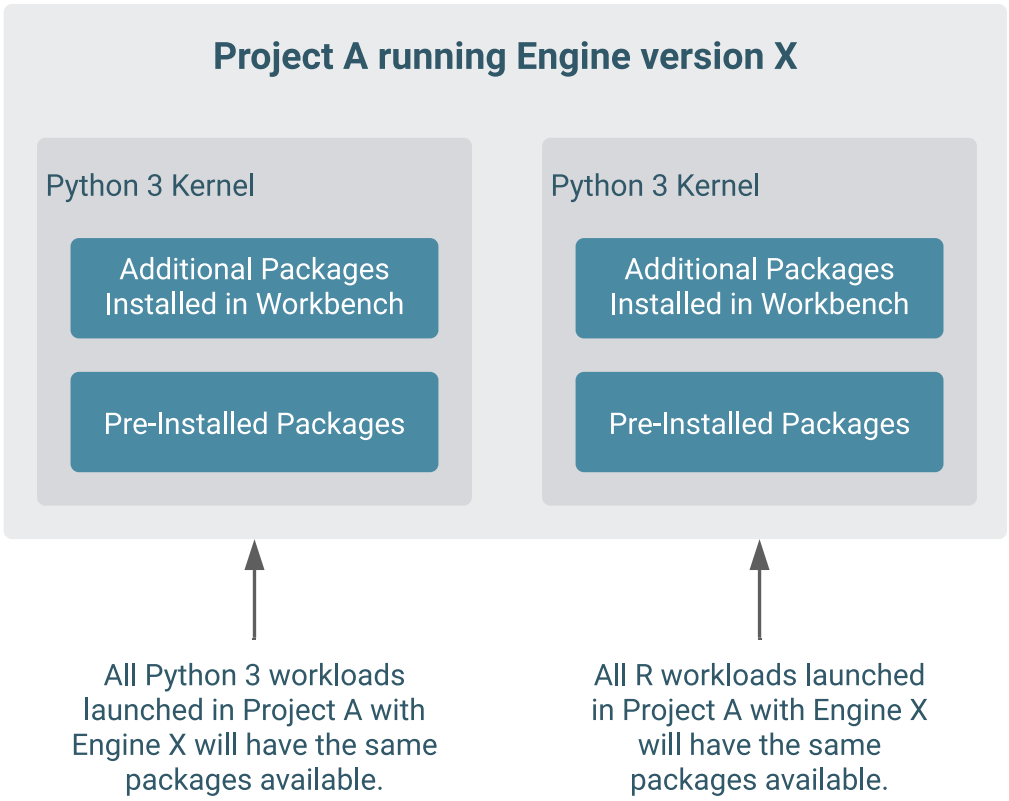
If you accidentally install these packages (or you see unexpected behavior when you switch a project from Legacy Engine to Runtimes), the simplest solution is to delete /home/cdsw/.local/lib/python and reinstall your project’s dependencies from the project overview page.

Engine Dependencies

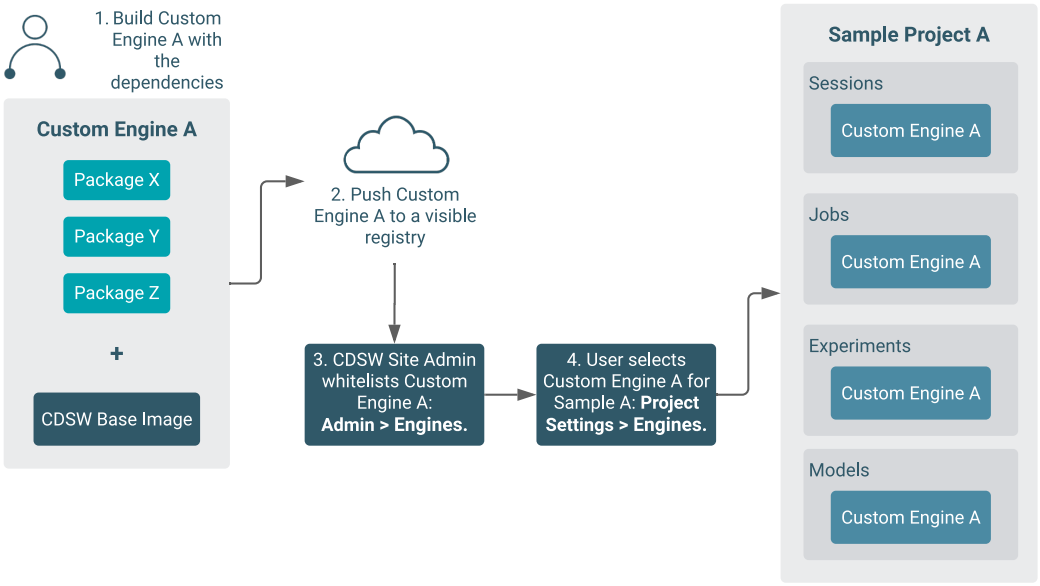
This topic describes the options available to you for mounting a project's dependencies into its engine environment. Depending on your projects or user preferences, one or more of these methods may be more appropriate for your deployment.

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*.

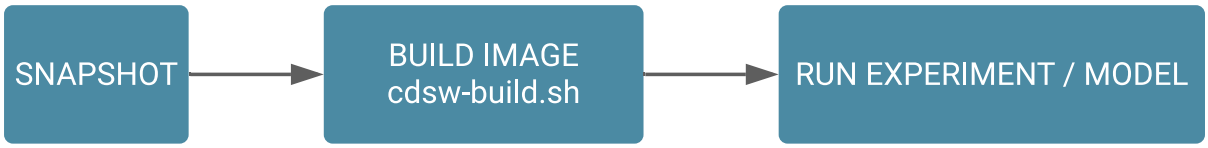
Related Information

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 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

Model Training and Deployment Overview

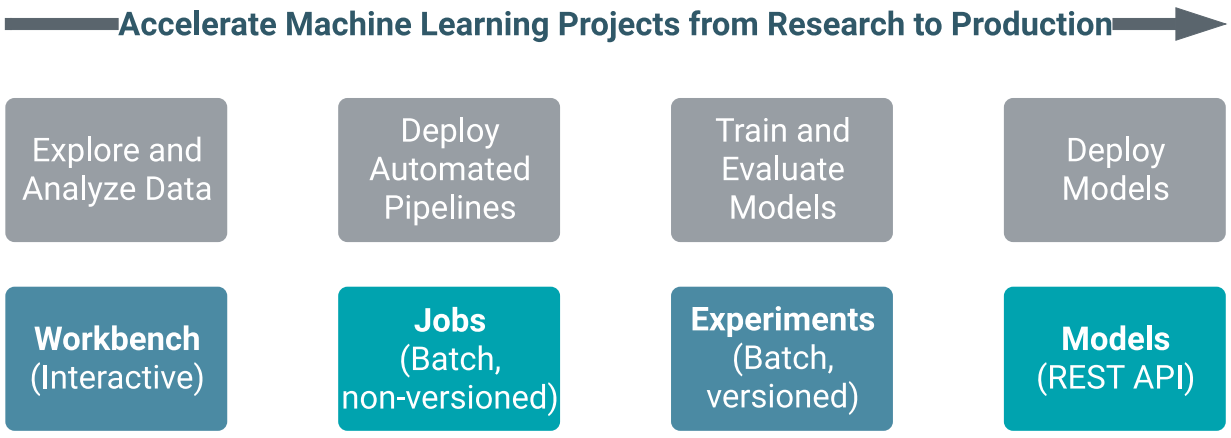
This section provides an overview of model training and deployment using Cloudera Machine Learning.

Machine learning is a discipline that uses computer algorithms to extract useful knowledge from data. There are many different types of machine learning algorithms, and each one works differently. In general however, machine learning algorithms begin with an initial hypothetical model, determine how well this model fits a set of data, and then work on improving the model iteratively. This training process continues until the algorithm can find no additional improvements, or until the user stops the process.

A typical machine learning project will include the following high-level steps that will transform a loose data hypothesis into a model that serves predictions.

1. Explore and experiment with and display findings of data
2. Deploy automated pipelines of analytics workloads
3. Train and evaluate models
4. Deploy models as REST APIs to serve predictions

With Cloudera Machine Learning, you can deploy the complete lifecycle of a machine learning project from research to deployment.



Experiments

This topic introduces you to experiments, and the challenge this feature aims to solve.

Cloudera Machine Learning allows data scientists to run batch experiments that track different versions of code, input parameters, and output (both metrics and files).

Challenge

As data scientists iteratively develop models, they often experiment with datasets, features, libraries, algorithms, and parameters. Even small changes can significantly impact the resulting model. This means data scientists need the ability to iterate and repeat similar experiments in parallel and on demand, as they rely on differences in output and scores to tune parameters until they obtain the best fit for the problem at hand. Such a training workflow requires versioning of the file system, input parameters, and output of each training run.

Without versioned experiments you would need intense process rigor to consistently track training artifacts (data, parameters, code, etc.), and even then it might be impossible to reproduce and explain a given result. This can lead to wasted time and effort during collaboration, not to mention the compliance risks introduced.

Solution

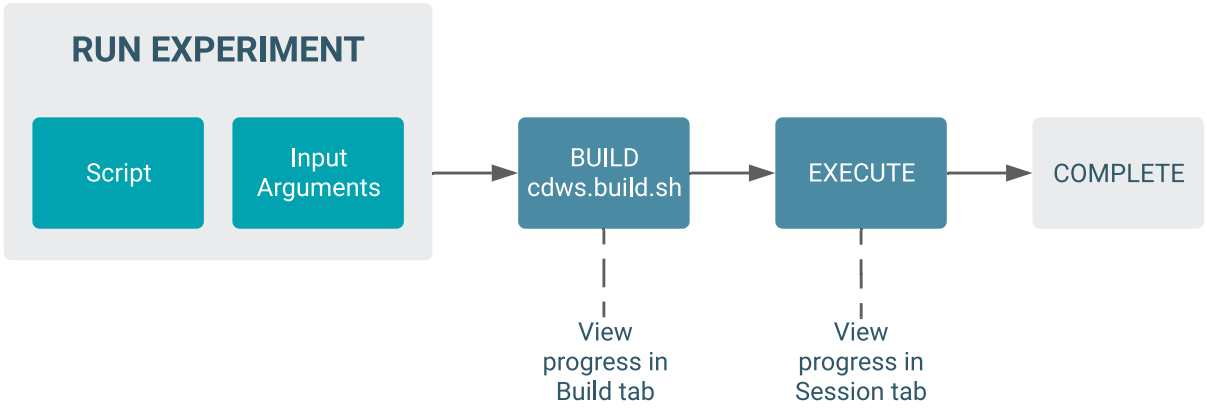
Cloudera Machine Learning uses experiments to facilitate ad-hoc batch execution and model training. Experiments are batch executed workloads where the code, input parameters, and output artifacts are versioned. This feature also provides a lightweight ability to track output data, including files, metrics, and metadata for comparison.

Experiments - Concepts and Terminology

This topic walks you through some basic concepts and terminology related to experiments.

The term experiment refers to a non interactive batch execution script that is versioned across input parameters, project files, and output. Batch experiments are associated with a specific project (much like sessions or jobs) and have no notion of scheduling; they run at creation time. To support versioning of the project files and retain run-level artifacts and metadata, each experiment is executed in an isolated container.

Lifecycle of an Experiment



The rest of this section describes the different stages in the lifecycle of an experiment - from launch to completion.

1. **Launch Experiment**

   In this step you will select a script from your project that will be run as part of the experiment, and the resources (memory/GPU) needed to run the experiment. The engine kernel will be selected by default based on your script. For detailed instructions on how to launch an experiment, see *Getting Started with Cloudera Machine Learning*.

2. **Build**

   When you launch the experiment, Cloudera Machine Learning first builds a new versioned engine image where the experiment will be executed in isolation. This new engine includes:

   - the base engine image used by the project (check Project Settings)
   - a snapshot of the project filesystem
   - environmental variables inherited from the project.
   - packages explicitly specified in the project's build script (cdsw-build.sh)

     It is your responsibility to provide the complete list of dependencies required for the experiment via the cdsw-build.sh file. As part of the engine's build process, Cloudera Machine Learning will run the cdsw-build.sh script and install the packages or libraries requested there on the new image.

   For details about the build process and examples on how to specify dependencies, see Engines for Experiments and Models. .

3. **Schedule**

   Once the engine is built the experiment is scheduled for execution like any other job or session. Once the requested CPU/GPU and memory have been allocated to the experiment, it will move on to the execution stage.

   Note that if your deployment is running low on memory and CPU, your runs may spend some time in this stage.

4. **Execute**

   This is the stage where the script you have selected will be run in the newly built engine environment. This is the same output you would see if you had executed the script in a session in the Workbench console.

   You can watch the execution in progress in the individual run's Session tab.

   You can also go to the project Overview Experiments page to see a table of all the experiments launched within that project and their current status.

   Run ID: A numeric ID that tracks all experiments launched on a Cloudera Machine Learning deployment. It is not limited to the scope of a single user or project.

Related Information

Running an Experiment with Cloudera Machine Learning

Models

Cloudera Machine Learning allows data scientists to build, deploy, and manage models as REST APIs to serve predictions.

Challenge

Data scientists often develop models using a variety of Python/R open source packages. The challenge lies in actually exposing those models to stakeholders who can test the model. In most organizations, the model deployment process will require assistance from a separate DevOps team who likely have their own policies about deploying new code.

For example, a model that has been developed in Python by data scientists might be rebuilt in another language by the devops team before it is actually deployed. This process can be slow and error-prone. It can take months to deploy new models, if at all. This also introduces compliance risks when you take into account the fact that the new re-developed model might not be even be an accurate reproduction of the original model.

Once a model has been deployed, you then need to ensure that the devops team has a way to rollback the model to a previous version if needed. This means the data science team also needs a reliable way to retain history of the models

they build and ensure that they can rebuild a specific version if needed. At any time, data scientists (or any other stakeholders) must have a way to accurately identify which version of a model is/was deployed.

Solution

Cloudera Machine Learning allows data scientists to build and deploy their own models as REST APIs. Data scientists can now select a Python or R function within a project file, and Cloudera Machine Learning will:

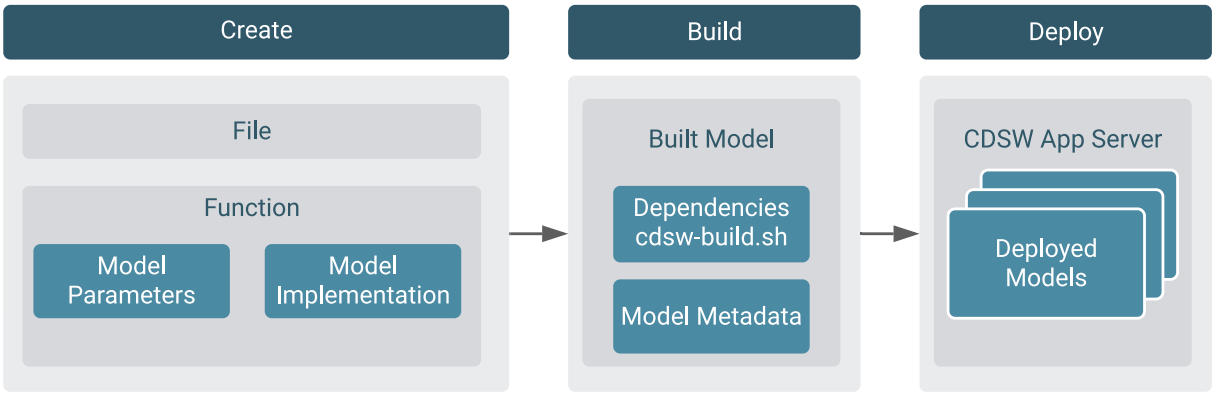
- Create a snapshot of model code, model parameters, and dependencies.
- Package a trained model into an immutable artifact and provide basic serving code.
- Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function's return type.
- Save the model along with some metadata.
- Deploy a specified number of model API replicas, automatically load balanced.

Models - Concepts and Terminology

Model

> Model is a high level abstract term that is used to describe several possible incarnations of objects created during the model deployment process. For the purpose of this discussion you should note that 'model' does not always refer to a specific artifact. More precise terms (as defined later in this section) should be used whenever possible.

Stages of the Model Deployment Process



The rest of this section contains supplemental information that describes the model deployment process in detail.

Create

- File - The R or Python file containing the function to be invoked when the model is started.
- Function - The function to be invoked inside the file. This function should take a single JSON-encoded object (for example, a python dictionary) as input and return a JSON-encodable object

as output to ensure compatibility with any application accessing the model using the API. JSON decoding and encoding for model input/output is built into Cloudera Machine Learning.

The function will likely include the following components:

- Model Implementation

  The code for implementing the model (e.g. decision trees, k-means). This might originate with the data scientist or might be provided by the engineering team. This code implements the model's predict function, along with any setup and teardown that may be required.
- Model Parameters

  A set of parameters obtained as a result of model training/fitting (using experiments). For example, a specific decision tree or the specific centroids of a k-means clustering, to be used to make a prediction.

Build

This stage takes as input the file that calls the function and returns an artifact that implements a single concrete model, referred to as a model build.

- Built Model

  A built model is a static, immutable artifact that includes the model implementation, its parameters, any runtime dependencies, and its metadata. If any of these components need to be changed, for example, code changes to the implementation or its parameters need to be retrained, a new build must be created for the model. Model builds are versioned using build numbers.

  To create the model build, Cloudera Machine Learning creates a Docker image based on the engine designated as the project's default engine. This image provides an isolated environment where the model implementation code will run.

  To configure the image environment, you can specify a list of dependencies to be installed in a build script called cdsw-build.sh.

  For details about the build process and examples on how to install dependencies, see *Engines for Experiments and Models*.
- Build Number:

  Build numbers are used to track different versions of builds within the scope of a single model. They start at 1 and are incremented with each new build created for the model.

Deploy

This stage takes as input the memory/CPU resources required to power the model, the number of replicas needed, and deploys the model build created in the previous stage to a REST API.

- Deployed Model

  A deployed model is a model build in execution. A built model is deployed in a model serving environment, likely with multiple replicas.
- Environmental Variable

  You can set environmental variables each time you deploy a model. Note that models also inherit any environment variables set at the project and global level. (For more information see *Engine Environment Variables*.) However, in case of any conflicts, variables set per-model will take precedence.

  Note: If you are using any model-specific environmental variables, these must be specified every time you re-deploy a model. Models do not inherit environmental variables from previous deployments.

- Model Replicas

  The engines that serve incoming requests to the model. Note that each replica can only process one request at a time. Multiple replicas are essential for load-balancing, fault tolerance, and serving concurrent requests. Cloudera Machine Learning allows you to deploy a maximum of 9 replicas per model.
- Deployment ID

  Deployment IDs are numeric IDs used to track models deployed across Cloudera Machine Learning. They are not bound to a model or project.

Related Information

Experiments - Concepts and Terminology

Engines for Experiments and Models

Engines Environment Variables

Collaborating on Projects with Cloudera Machine Learning

This topic discusses all the collaboration strategies available to Cloudera Machine Learning users.

Project Collaborators

If you want to work closely with trusted colleagues on a particular project, you can add them to the project as collaborators. This is recommended for collaboration over projects created under your personal account. Anyone who belongs to your organization can be added as a project collaborator.

Project Visibility Levels: When you create a project in your personal context, Cloudera Machine Learning asks you to assign one of the following visibility levels to the project - Private or Public. Public projects on Cloudera Machine Learning grant read-level access to everyone with access to the Cloudera Machine Learning application. For Private projects, you must explicitly add someone as a project collaborator to grant them access.

Project Collaborator Access Levels: You can grant project collaborators the following levels of access: Viewer, Operator, Contributor, Admin

Note:

Collaborating Securely on Projects

Before adding project collaborators, you must remember that assigning the Contributor or Admin role to a project collaborator is the same as giving them write access to your data in CDH. This is because project contributors and project administrators have write access to all your project code (including any library code that you might not be actively inspecting). For example, a contributor/admin could modify project file(s) to insert code that deletes some data on the cluster. The next time you launch a session and run the same code, it will appear as though you deleted the data yourself.

Additionally, project collaborators also have access to all actively running sessions and jobs by default. This means that a malicious user can easily impersonate you by accessing one of your active sessions. Therefore, it is extremely important to restrict project access to trusted collaborators only. Note that site administrators can restrict this ability by allowing only session creators to run commands within their own active sessions.

For these reasons, Cloudera recommends using Git to collaborate securely on shared projects.

Teams

Users who work together on more than one project and want to facilitate collaboration can create a Team. Teams allow streamlined administration of projects. Team projects are owned by the team, rather than an individual user. Only users that are already part of the team can be added as collaborators to projects created within the team context. Team administrators can add or remove members at any time, assigning each member different access permissions.

Team Member Access Levels: You can grant team members the following levels of access: Viewer, Operator, Contributor, Admin.

ML Business User

The ML Business User role is for a user who only needs to view any applications that are created within Cloudera Machine Learning. This is the ideal role for an employee who is not part of the Data Science team and does not need higher-level access to workspaces and projects, but needs to access the output of a Data Science workflow. MLBusinessUser seats are available for purchase separately.

Forking Projects

You can fork another user's project by clicking Fork on the Project page. Forking creates a new project under your account that contains all the files, libraries, configurations, jobs, and dependencies between jobs from the original project.

Creating sample projects that other users can fork helps to bootstrap new projects and encourage common conventions.

Collaborating with Git

Cloudera Machine Learning provides seamless access to Git projects. Whether you are working independently, or as part of a team, you can leverage all of benefits of version control and collaboration with Git from within Cloudera Machine Learning. Teams that already use Git for collaboration can continue to do so. Each team member will need to create a separate Cloudera Machine Learning project from the central Git repository.

For anything but simple projects, Cloudera recommends using Git for version control. You should work on Cloudera Machine Learning the same way you would work locally, and for most data scientists and developers that means using Git.

Sharing Job and Session Console Outputs

This topic describes how to share the results of your research (that is, output from sessions and jobs) with teammates and project stakeholders.

Cloudera Machine Learning lets you easily share the results of your analysis with one click. Using rich visualizations and documentation comments, you can arrange your console log so that it is a readable record of your analysis and results. This log continues to be available even after the session stops. This method of sharing allows you to show colleagues and collaborators your progress without your having to spend time creating a report.

To share results from an interactive session, click Share at the top of the console page. From here you can generate a link that includes a secret token that gives access to that particular console output. For jobs results, you can either share a link to the latest job result or a particular job run. To share the latest job result, click the Latest Run link for a job on the Overview page. This link will always have the latest job results. To share a particular run, click on a job run in the job's History page and share the corresponding link.

You can share console outputs with one of the following sets of users.

- All anonymous users with the link - By default, Cloudera Machine Learning allows anonymous access to shared consoles. However, site administrators can disable anonymous sharing at any time.

  Once anonymous sharing has been disabled, all existing publicly shared console outputs will be updated to be viewable only by authenticated users.
- All authenticated users with the link - This means any user with a Cloudera Machine Learning account will have access to the shared console.
- Specific users and teams - Click Change to search for users and teams to give access to the shared console. You can also come back to the session and revoke access from a user or team the same way.

Sharing Data Visualizations

If you want to share a single data visualization rather than an entire console, you can embed it in another web page. Click the small circular 'link' button located to the left of most rich visualizations to view the HTML snippet that you can use to embed the visualization.

Autoscaling Workloads with Kubernetes

Autoscaling on Private Cloud

CML on Private Cloud supports application autoscaling on multiple fronts. Additional compute resources are utilized when users self-provision sessions, run jobs, and utilize other compute capabilities. Within a session, users can also leverage the worker API to launch resources necessary to host TensorFlow, PyTorch, or other distributed applications. Spark on Kubernetes scales up to any number of executors as requested by the user at runtime.

Planning

The information in this section will help you plan your Cloudera Machine Learning installation. advance

Introduction to Private Cloud

With the Cloudera Machine Learning (CML) service, data scientists and partners can build and run machine learning experiments and workloads in a secure environment. CML on Private Cloud provides an identical experience to CML on Public Cloud, but running in your own on-premises data center.

Cloudera Machine Learning enables you to:

- Easily onboard a new tenant and provision an ML workspace in a shared OpenShift or ECS environment.
- Enable data scientists to access shared data on CDP Private Cloud Base and CDW.
- Leverage Spark-on-K8s to spin up and down Spark clusters on demand.

Cloudera Machine Learning requirements (OCP)

To launch the Cloudera Machine Learning service, the OpenShift Container Platform (OCP) host must meet several requirements. Review the following CML-specific software, NFS server, and storage requirements.

Requirements

If necessary, contact your Administrator to make sure the following requirements are satisfied:

1. If you are using OpenShift, the installed OpenShift Container Platform must be version 4.7 or 4.8. For ECS, refer to the Hardware and Software Requirements section in Installing and Managing a Private Cloud Experience Cluster 1.4.0.
2. CML assumes it has cluster-admin privileges on the cluster.
3. Storage:
   a. 4 TB of persistent volume block storage per ML Workspace.
   b. 1 TB of NFS space recommended per Workspace (depending on user files).
   c. Access to NFS storage is routable from all pods running in the cluster.
   d. For monitoring, recommended volume size is 60 GB.

4. A block storage class must be marked as default in the cluster. This may be rook-ceph-block, Portworx, or another storage system. Confirm the storage class by listing the storage classes (run `oc get sc`) in the cluster, and check that one of them is marked default.
5. If external NFS is used, the NFS directory and assumed permissions must be those of the cdsw user. For details see Using an External NFS Server in the Related information section at the bottom of this page.
6. If CML needs access to a database on the CDP Private Cloud Base cluster, then the user must be authenticated using Kerberos and must have Ranger policies set up to allow read/write operations to the default (or other specified) database.
7. Ensure that Kerberos is enabled for all services in the cluster. Custom Kerberos principals are not currently supported. For more information, see Enabling Kerberos for authentication.
8. Forward and reverse DNS must be working.
9. DNS lookups to sub-domains and the ML Workspace itself should work.
10. In DNS, wildcard subdomains (such as *.cml.yourcompany.com) must be set to resolve to the master domain (such as cml.yourcompany.com). The TLS certificate (if TLS is used) must also include the wildcard subdomains. When a session or job is started, an engine is created for it, and the engine is assigned to a random, unique subdomain.
11. The external load balancer server timeout needs to be set to 5 min. Without this, creating a project in an ML workspace with `git clone` or with the API may result in API timeout errors. For workarounds, see Known Issue DSE-11837.
12. If you intend to access a workspace over https, see Deploy an ML Workspace with Support for TLS.
13. For non-TLS ML workspaces, websockets need to be allowed for port 80 on the external load balancer.
14. Only a TLS-enabled custom Docker Registry is supported. Ensure that you use a TLS certificate to secure the custom Docker Registry. The TLS certificate can be self-signed, or signed by a private or public trusted Certificate Authority (CA).
15. On OpenShift, due to a Red Hat issue with OpenShift Container Platform 4.3.x, the image registry cluster operator configuration must be set to Managed.
16. Check if storage is set up in the cluster image registry operator. See Known Issues DSE-12778 for further information.

For more information on requirements, see CDP Private Cloud Base Installation Guide.

Hardware requirements

Storage

The cluster must have persistent storage classes defined for both block and filesystem volumeModes of storage. Ensure that a block storage class is set up. The exact amount of storage classified as block or filesystem storage depends on the specific workload used:

- Machine Learning workload requirements for storage largely depend on the nature of your machine learning jobs. 4 TB of persistent volume block storage is required per Machine Learning Workspace instance for storing different kinds of metadata related to workspace configuration. Additionally, Machine Learning requires access to NFS storage routable from all pods running in the cluster (see below).
- Monitoring uses a large Prometheus instance to scrape workloads. Disk usage depends on scale of workloads. Recommended volume size is 60 GB.

	Local Storage (for example, ext4)	Block PV (for example, Ceph or Portworx)	NFS (for ML user project files)
Control Plane	N/A	250 GB	N/A
CML	N/A	4 TB per workspace	1 TB per workspace (dependent on size of ML user files)

NFS

Cloudera Machine Learning (CML) requires NFS 4.0 for storing project files and folders. NFS storage is to be used only for storing project files and folders, and not for any other CML data, such as PostgreSQL database and LiveLog.

ECS requirements for NFS Storage

Cloudera managed ECS deploys and manages an internal NFS server based on LongHorn which can be used for CML. This is the recommended option for CML on ECS clusters. CML requires nfs-utils in order to mount longhorn-nfs provisioned mounts.

CML requires the nfs-utils package be installed in order to mount volumes provisioned by longhorn-nfs. The nfs-utils package is not available by default on every operating system. Check if nfs-utils is available, and ensure that it is present on all ECS cluster nodes.

Alternatively, the NFS server can be external to the cluster, such as a NetApp filer that is accessible from the private cloud cluster nodes.

OpenShift requirements for NFS storage

An internal user-space NFS server can be deployed into the cluster which serves a block storage device (persistent volume) managed by the cluster's software defined storage (SDS) system, such as Ceph or Portworx. This is the recommended option for CML on OpenShift. Alternatively, the NFS server can be external to the cluster, such as a NetApp filer that is accessible from the private cloud cluster nodes. NFS storage is to be used only for storing project files and folders, and not for any other CML data, such as PostgreSQL database and LiveLog.

CML does not support shared volumes, such as Portworx shared volumes, for storing project files. A read-write-once (RWO) persistent volume must be allocated to the internal NFS server (for example, NFS server provisioner) as the persistence layer. The NFS server uses the volume to dynamically provision read-write-many (RWX) NFS volumes for the CML clients.

Related Information

CDP Private Cloud Base Installation Guide

CDP Private Cloud Experiences Installation Software Requirements

Known Issues and Limitations

Deploy an ML Workspace with Support for TLS

Using an External NFS Server

Cloudera Machine Learning requirements (ECS)

There are minimal requirements when using Cloudera Machine Learning on ECS.

The primary requirement is to have 4 TB of storage space.

For further information, see the Hardware and Software Requirements section in *CDP Private Cloud Data Services installation overview*

Related Information

CDP Private Cloud Data Services installation overview

Get started with CML on Private Cloud

To get started as a user with Cloudera Machine Learning on your Private Cloud, follow the steps described below. They will show you how to set up a Project and work on some data.

Before you begin

Make sure the Admin creates a new Workspace for you. If you are an Admin, see: Provision an ML Workspace.

Note: Make sure that an Admin user logs into the Workspace first.

Procedure

1. Log in to your workspace. On the Workspaces tab, click Launch Workspace.
2. Next, create a Project. See: Creating a Project.
3. Once you have a Project, run a Session to start your work. See: Launch a Session.
4. Test your access to the base cluster (Data Lake). See: CDP-DC cluster connectivity test.
5. You can then run a Model. Learn about Models here: Creating and Deploying a Model.
6. When you are finished with your workspace, your Admin can remove it, as described here: Removing ML Workspaces.

Test your connectivity to the CDP-DC cluster

Test that you can create a Project in your ML Workspace and access data that is stored in the data center cluster.

Procedure

1. Create a new Project, using the PySpark template.
2. Create a new file called testdata.txt (use this exact filename).
3. Add 2-3 lines of any text in the file to serve as sample data.
4. Run the following Spark commands to test the connection.

```
from pyspark.sql import SparkSession

# Instantiate Spark-on-K8s Cluster
spark = SparkSession\
.builder\
.appName("Simple Spark Test")\
.config("spark.executor.memory","8g")\
.config("spark.executor.cores","2")\
.config("spark.driver.memory","2g")\
.config("spark.executor.instances","2")\
.getOrCreate()

# Validate Spark Connectivity
spark.sql("SHOW databases").show()
spark.sql('create table testcml (abc integer)').show()
spark.sql('insert into table testcml  select t.* from (select 1) t').show(
)
spark.sql('select * from testcml').show()
# Stop Spark Session
spark.stop()
```

5. Run the following direct HDFS commands to test the connection.

```
# Run sample HDFS commands
# Requires an additional testdata.txt file to be created with sample data
 in project home dir
!hdfs dfs -mkdir /tmp/testcml/
!hdfs dfs -copyFromLocal /home/cdsw/testdata.txt /tmp/testcml/
!hdfs dfs -cat /tmp/testcml/testdata.txt
```

What to do next

If you get errors, then check with your Admin to make sure that your user ID is set up in the Hadoop Authentication settings to access the CDP-DC cluster, and that the correct Ranger permissions have been applied.

Differences Between Public and Private Cloud

There are some differences in Cloudera Machine Learning functionality between Public and Private Cloud.

Feature	Public Cloud	Private Cloud 1.4.x
CML application control plane (infrastructure containers and workload containers)	Control plane is hosted on public cloud servers.	Control plane is hosted on customer's cluster.
Storage - CML internal state data (database, images, logs)	EBS on AWS, Azure Disks on Azure.	Software Defined Storage System, such as Ceph or Portworx.
Storage - User project files	EFS on AWS, external NFS on Azure.	Internal NFS storage is recommended.
Autoscaling	CPU/GPU nodes scale up and down as needed.	Autoscaling concept is different; Private Cloud shares a pooled set of resources among workloads.
Logging	Per-workspace diagnostic bundles can be downloaded from the workspace.	Diagnostic bundles are not supported at Workspace level, but can be downloaded from the control plane at the cluster level.
Monitoring dashboards	Provides four dashboards.	Provides two dashboards, for K8s Container and K8s Cluster.
NFS support	AWS uses EFS; Azure requires external NFS.	Internal NFS is recommended, external NFS is supported.
TLS support	TLS access to workspaces is supported.	TLS access is supported, but requires manual setup of certificate and other steps.
Hadoop Authentication	Uses FreeIPA	User needs to provide credentials to communicate with the CDP Private Base cluster.
Remote Access	Available from each workspace.	Not available in the workspace. Instead, the environment's kubeconfig file may be downloaded from Environments using the Download Kubernetes configuration action for the specified environment.
Roles	MLAdmin, MLUser	The corresponding roles are: EnvironmentAdmin, EnvironmentUser

Limitations on Private Cloud

There are some limitations to keep in mind when you are working with Cloudera Machine Learning on Private Cloud.

The following features are not yet supported in CML Private Cloud:

- Logging is limited, and diagnostic bundles for each workspace cannot be downloaded from the workspace UI. Instead, diagnostic bundles for the entire cluster can be downloaded from the control plane.
- Monitoring on Private Cloud does not support node-level resource metrics, hence only K8s Cluster and K8s Container dashboards are available.
- CML does not support the NVIDIA Multi-Instance GPU (MIG) feature.

Network File System (NFS)

A Network File System (NFS) is a protocol to access storage on a network that emulates accessing storage in a local file system. CML requires an NFS server for storing project files and folders, and the NFS export must be configured before you provision the first CML workspace in the cluster.

There are many different products or packages that can create an NFS in your private network. A Kubernetes cluster can host an internal NFS server, or an external NFS server can be installed on another cluster that is accessible by the

private cloud cluster nodes. NFS storage is used only for storing project files and folders, and not for any other CML data, such as PostgreSQL database and livelog files.

CML does not support shared volumes, such as Portworx shared volumes, for storing project files. A read-write-once (RWO) persistent volume must be allocated to the internal NFS server (for example, NFS server provisioner) as the persistence layer. The NFS server uses the volume to dynamically provision read-write-many (RWX) NFS volumes for the CML clients.

An external NFS server option is currently the recommended option for Private Cloud production workloads. Not specifying an external NFS Server for your ML Workspace will use/require a deprecated internal NFS provisioner, which should only be used for small, proof-of-concept deployments. There are several options for setting up an internal NFS provisioner, described in the appendix. The Private Cloud Admin is responsible for setting up an NFS for use by your cluster.

Note: See *CDP Private Cloud Data Services Installation Software Requirements* for some information about installing NFS.

Related Information

CDP Private Cloud Experiences Installation Software Requirements

NFS Options for Private Cloud

Cloudera Machine Learning on Private Cloud requires a Network File System (NFS) server for storing project files and folders.

On ECS, NFS is part of the overall installation, and no additional setup steps are required.

To use an NFS server that is external to the cluster, such as a NetApp Filer appliance. In this case, you must manually create a directory for each workspace. In this case, the NFS server must be configured before deploying the first CML workspace in the cluster. One important limitation is that CML does not support using shared volumes for storing project files. A read-write-once (RWO) persistent volume must be allocated to the internal NFS server (e.g., NFS server provisioner) as the persistence layer. The NFS server uses the volume to dynamically provision read-write-many (RWX) NFS volumes for the CML clients.

An alternative approach is to use an internal NFS server which is deployed into the cluster. This method uses a deprecated internal NFS provisioner, and it should only be used for small, proof-of-concept deployments. Solutions include NFS over Ceph using NFS Server Provisioner (NFS Ganesha) on OpenShift. On ECS, Cloudera manages and deploys an NFS which can be used for CML The storage space for each workspace is transparently managed by the internal NFS server.

Internal Network File System on OCP (Non-Production)

Learn about deploying, backing up, and uninstalling an internal NFS server on OpenShift Container Platform. However, for production, an external NFS server is recommended. For more information, see *Using an External NFS Server*.

Deploying NFS Server Provisioner on Rook Ceph

As an example, you can deploy the NFS Server Provisioner using the Helm chart provided here.

- For the nfs storage class, set “allowVolumeExpansion=true”
- For the underlying persistent volume, specify a size of 1 TiB.
- On the block storage system class, rook-ceph-block in this case, set allowVolumeExpansion=true
- Download two yaml files here: nfs-server-provisioner.yml and nfs-scc.yml.

Note: If your read-write-once block pv storage class is not rook-ceph-block, then replace rook-ceph-block with your preferred read-write-once block pv storage class in the nfs-server-provisioner.yml file below.

Note: In this example, the size of the underlying persistent volume is specified as 1 TiB. You are recommended to increase the size by 1 TiB for each additional ML workspace that will be created. For example, increase the size by 5 TiB for 5 ML workspaces.

Note: The yaml file below has a line that will download the nfs server image from quay.io/kubernetes_incubator/nfs-provisioner:v2.3.0. If your installation is airgapped, download the file to a local repository and change the path accordingly.

1. Install Path 1: Installing using the oc command and yaml files:
   a. If you do not have Tillerless Helm v2 set up, then you can simply apply the nfs-server-provisioner.yml file as follows: $ oc create -f nfs-server-provisioner.yml -n cml-nfs
2. Install Path 2: Installing using the oc command and Tillerless Helm v2:

```
$ oc delete scc nfs-scc
       $ oc delete clusterrole cml-nfs-nfs-server-provisioner
       $ oc delete clusterrolebinding cml-nfs-nfs-server-provisioner
       $ oc delete namespace cml-nfs

       $ helm tiller run cml-nfs -- helm delete cml-nfs --purge
       $ oc delete scc nfs-scc securitycontextconstraints.security.openshi
ft.io "nfs-scc" deleted
```

Backing up Project Files and Folders

The block device backing the NFS server data must be backed up to protect the CML project files and folders. The backup mechanism would vary depending on the underlying block storage system and backup policies in place.

1. identify the underlying block storage to backup, first determine the NFS PV:

```
$ echo `kubectl get pvc -n cml-nfs -o jsonpath='{.items[0].spec.volumeNa
me}'`
       pvc-bec1de27-753d-11ea-a287-4cd98f578292
```

2. For Rook Ceph, the RBD volume/image name is the name of the dynamically created persistent volume (pvc-3d3316b6-6cc7-11ea-828e-1418774847a1).

Ensure this volume is backed up using an appropriate backup policy.

Uninstalling the NFS server on OpenShift

Uninstall the NFS server provisioner using either of the following commands.

Use this command if the NFS server provisioner was installed using oc and yaml files:

```
$ oc delete scc nfs-scc
$ oc delete clusterrole cml-nfs-nfs-server-provisioner
$ oc delete clusterrolebinding cml-nfs-nfs-server-provisioner
$ oc delete namespace cml-nfs
```

Use this command if the NFS server provisioner was installed using `Helm`:

```
$ helm tiller run cml-nfs -- helm delete cml-nfs --purge
$ oc delete scc nfs-scc securitycontextconstraints.security.openshift.io "n
fs-scc" deleted
```

Related Information

Using an External NFS Server

Internal Network File System on ECS (Non-Production)

On ECS, NFS is part of the overall installation, and no additional setup steps are required. However, for production, an external NFS server is recommended. For more information, see *Using an External NFS Server*.

The internal NFS does not have a backup feature.

Related Information

Using an External NFS Server

Using an External NFS Server

You can install an NFS server that is external to the cluster.

About this task

Currently, NFS version 4.1 is the recommended protocol to use for CML. The NFS client within CML must be able to mount the NFS storage with default options, and also assumes these export options:

```
rw,sync,no_root_squash,no_all_squash,no_subtree_check
```

Before you begin

Before creating a CML workspace, the storage administrator must create a directory that will be exported to the cluster for storing ML project files for that workspace. Either a dedicated NFS export path, or a subdirectory in an existing export must be specified for each workspace.

Each CML workspace needs a unique directory that does not have files in it from a different or previous workspace. For example, if 10 CML workspaces are expected, the storage administrator will need to create 10 unique directories. Either one NFS export and 10 subdirectories within it need to be created, or 10 unique exports need to be created.

For example, to use a dedicated NFS share for a workspace named “workspace1” from NFS server “nfs_server”, do the following:

Procedure

1. Create NFS export directory “/workspace1”.
2. Change ownership for the exported directory
   a) CML accesses this directory as a user with a UID and GID of 8536. Therefore, run chown 8536:8536 /workspace1
   b) Make the export directory group-writeable and set the GID:
   chmod g+srwx /workspace1
3. Provide the NFS export path nfs_server:/workspace1 when prompted by the CML Control Plane App while creating the workspace.
4. To use a subdirectory in an existing NFS share, say nfs_server:/export, do the following:
   a) Create a subdirectory /export/workspace1
   b) Change ownership: chown 8536:8536 /export/workspace1
   c) Set GID and make directory group writeable: chmod g+srwx /export/workspace1
   d) Provide the export path nfs_server:/export/workspace1 when prompted by the CML Control Plane App.

NFS share sizing

CML workloads are sensitive to latency and IO/s instead of throughput.

The minimum recommended file share size is 100 GB. The file share must support online volume capacity expansion. It must provide at least the following performance characteristics:

IO / s	3100
Throughput rate	110.0 MiBytes/s

Deploy an ML Workspace with Support for TLS

You can provision an ML workspace with TLS enabled, so that it can be accessed via https.

Before you begin

You need to obtain a certificate from the Certificate Authority used by your organization. This may be an internal certificate authority.

Additionally, you need a computer with CLI access to the cluster, and with kubectl installed.

Procedure

1. Provision the ML Workspace. Follow the procedure Provisioning ML Workspaces.

   Note: Ensure you select Enable TLS.

2. Obtain the .crt and .key files for the certificate from your Certificate Authority.

   The certificate URL is generally of the form: <workspaceid>.<cluster>.<domain>.com. Assuming an example URL for the certificate of ml-30b43418-53c.cluster.yourcompany.com, check that the certificate correctly shows the corresponding Common Name (CN) and Subject Alternative Names (SAN):

   - CN: ml-30b43418-53c.cluster.yourcompany.com
   - SAN: *.ml-30b43418-53c.cluster.yourcompany.com
   - SAN: ml-30b43418-53c.cluster.yourcompany.com

3. Create a Kubernetes secret inside the previously provisioned ML workspace namespace, and name the secret cml-tls-secret.

   On a machine with access to the .srt and .key files above, and access to the OpenShift cluster, run this command: kubectl create secret tls cml-tls-secret --cert=<pathtocrt.crt> --key=<pathtokey.key> -o yaml --dry-run | kubectl -n <cml-workspace-namespace> create -f -

   You can replace or update certificates in the secret at any time.

4. In Admin Security Root CA configuration , add the root CA certificate to the workspace.

   For example: https://ml-def88113-acd.cluster.yourcompany.com/administration/security"

Results

The command creates routes to reflect the new state of ingress and secret, and enables TLS.

Replace a Certificate

You can replace a certificate in a deployed namespace.

About this task

Procedure

1. Obtain the new certificate .crt and .key files.
2. Run this command (example): kubectl create secret tls cml-tls-secret --cert=<pathtocrt.crt> --key=<pathtokey .key> -o yaml --dry-run | kubectl -n <cml-workspace-namespace> replace -f -

What to do next

The certificate of an existing session does not get renewed. The new certificate only applies to newly created sessions.

Deploy an ML Workspace with Support for TLS on ECS

On ECS, you can provision an ML workspace with TLS enabled, so that the workspace is accessible via https.

About this task

You need to obtain a certificate from the Certificate Authority used by your organization. This may be an internal certificate authority. Additionally, you need a computer with CLI access to the cluster, and with `kubectl` installed.

Procedure

1. Provision an ML workspace. See *Provision an ML Workspace* for more information.

   Note: Ensure you select Enable TLS.

2. Obtain the .crt and .key files for the certificate from your Certificate Authority.

   The certificate URL is generally of the form: <workspaceid>.<cluster>.<domain>.com. Assuming an example URL for the certificate of ml-30b43418-53c.cluster.yourcompany.com, check that the certificate correctly shows the corresponding Common Name (CN) and Subject Alternative Names (SAN):

   - CN: ml-30b43418-53c.cluster.yourcompany.com
   - SAN: *.ml-30b43418-53c.cluster.yourcompany.com
   - SAN: ml-30b43418-53c.cluster.yourcompany.com

3. Create or replace a Kubernetes secret inside the previously provisioned ML workspace namespace, then automatically upload the certificate.

   Login to the Ecs Server role host and execute the following commands to accomplish these steps:

   a) cd /opt/cloudera/parcels/ECS/bin/

   b) ./cml_utils.sh -h

      Optional: A helper prompt appears, with explanation for the next command.

   c) ./cml_utils.sh upload-cert -n <namespace> -c <path_to_cert> -k <path_to_key>

      For example: ./cml_utils.sh upload-cert -n bb-tls-1 -c /tmp/ws-cert.crt -k /tmp/ws-key.key

      Note: To find the <namespace> of the workspace, go to the Machine Learning Workspaces UI, and in the Actions menu for the workspace, select View Workspace Details. Namespace is shown on the Details tab.

4. In Site Administration Security Root CA configuration , add the root CA certificate to the workspace.

   For example: https://ml-def88113-acd.apps.nf-01.os4cluster.yourcompany.com/administration/security

GPU node setup

In Kubernetes, you can taint nodes to affect how the node is scheduled. You can ensure that nodes that have a GPU are reserved exclusively for CML workloads that require a GPU.

To reserve a GPU node, assign a taint to the node.

Openshift

On Openshift, specify the node taint nvidia.com/gpu: true:NoSchedule for any nodes that host GPUs and are required to be used only for GPU workloads.

ECS

On ECS, set the node taint nvidia.com/gpu: true:NoSchedule in one of the following three ways:

1. During ECS installation: After adding the GPU host(s) to Cloudera Manager but prior to creation of the ECS cluster, visit the Host Configuration page, select the Dedicated GPU Node for Data Services checkbox and Save the configuration. Repeat for all hosts on which the taint is desired. Then, proceed with installation via the Add Cluster wizard.
2. During ECS upgrade: After upgrading Cloudera Manager (if applicable), set the host configuration as described above on one or more hosts in the ECS cluster. Then, proceed with upgrade via the Upgrade Cluster wizard.
3. Independently of ECS install or upgrade: Set the host configuration as described above on one or more hosts in the ECS cluster. Redeploy the client configuration on the ECS cluster. Finally, run the Reapply All Settings to Cluster command on the ECS service, which can be found in the Service Actions menu.

How To

you can learn about the various features and functions of Cloudera Machine Learning in the following sections.

Provision an ML Workspace

In CML on Private Cloud, the ML Workspace is where data scientists get their work done. After your Admin has created or given you access to an environment, you can set up a workspace. Only one workspace can be created per environment.

Before you begin

The first user to access the ML workspace after it is created must have the EnvironmentAdmin role assigned.

Procedure

1. Log in to the CDP Private Cloud web interface using your corporate credentials or other credentials that you received from your CDP administrator.
2. Click ML Workspaces.
3. Click Provision Workspace. The Provision Workspace panel displays.
4. In Provision Workspace, fill out the following fields.
   a) Workspace Name - Give the ML workspace a name. For example, test-cml.
   b) Select Environment - From the dropdown, select the environment where the ML workspace must be provisioned. If you do not have any environments available to you in the dropdown, contact your CDP admin to gain access.
   c) Namespace - Enter the namespace to use for the ML workspace.
   d) NFS Server - Select Internal to use an NFS server that is integrated into the Kubernetes cluster. This is the recommended selection at this time.

      The path to the internal NFS server is already set in the environment.
5. In Production Machine Learning, select to enable the following features.
   a) Enable Governance - Enables advanced lineage and governance features.

      Governance Principal Name - If Enable Governance is selected, you can use the default value of mlgov, or enter an alternative name. The alternative name must be present in your environment and be given permissions in Ranger to allow the MLGovernance service deliver events to Atlas.
   b) Enable Model Metrics - Enables exporting metrics for models to a PostgreSQL database.
6. In Other Settings, select to enable the following features.
   a) Enable TLS - Select this to enable https access to the workspace.
   b) Enable Monitoring - Administrators (users with the EnvironmentAdmin role) can use a Grafana dashboard to monitor resource usage in the provisioned workspace.
7. Click Provision Workspace. The new workspace provisioning process takes several minutes.

What to do next

After the workspace is provisioned, you can log in by clicking the workspace name on the Machine Learning Workspaces page. The first user to log in must be the administrator.

Related Information

Monitoring ML Workspaces

Removing ML Workspaces

Monitoring ML Workspaces

This topic shows you how to monitor resource usage on your ML workspaces.

About this task

Cloudera Machine Learning leverages Prometheus and Grafana to provide a dashboard that allows you to monitor how CPU, memory, storage, and other resources are being consumed by ML workspaces. Prometheus is an internal data source that is auto-populated with resource consumption data for each workspace. Grafana is a monitoring dashboard that allows you to create visualizations for resource consumption data from Prometheus.

Each ML workspace has its own Grafana dashboard.

Before you begin

Required Role: MLAdmin

You need the MLAdmin role to view the Workspace details page.

Note: On Private Cloud, the corresponding role is EnvironmentAdmin.

Procedure

1. Log in to the CDP web interface.
2. Click ML Workspaces.
3. For the workspace you want to monitor, click Actions Open Grafana .

Results

CML provides you with several default Grafana dashboards:

- K8s Cluster: Shows cluster health, deployments, and pods
- K8s Containers: Shows pod info, cpu and memory usage
- K8s Node: Shows node cpu and memory usage, disk usage and network conditions
- Models: Shows response times, requests per second, cpu and memory usage for model replicas.

You might choose to add new dashboards or create more panels for other metrics. For more information, see the *Grafana documentation*.

Related Information

Monitoring and Alerts

Removing ML Workspaces

This topic describes how to remove an existing ML workspace and clean up any cloud resources associated with the workspace. Currently, only CDP users with both the MLAdmin role and the EnvironmentAdmin account role can remove workspaces.

Procedure

1. Log in to the CDP web interface.
2. Click ML Workspaces.
3. Click on the Actions icon and select Remove Workspace.
   a) Force Delete - This property is not required by default. You should first attempt to remove your workspace with this property disabled.

      Enabling this property deletes the workspace from CDP but does not guarantee that the underlying kubernetes resources used by the workspace are cleaned up properly. Go to you kuberknetes administration console to make sure that the resources have been successfully deleted.
4. Click OK to confirm.

User Roles

Users in Cloudera Machine Learning are assigned one or more of the following roles.

There are two categories of roles: environment resource roles, which apply to a given CDP environment, and workspace resource roles, which apply to a single workspace. To use workspace resource roles, you may need to upgrade the workspace or create a new workspace.

If a user has more than one role, then the role with the highest level of permissions takes precedence. If a user is a member of a group, it may gain additional roles through that membership.

Environment resource roles

- MLAdmin: Grants a CDP user the ability to create and delete Cloudera Machine Learning workspaces within a given CDP environment. MLAdmins also have Administrator level access to all the workspaces provisioned within this environment. They can run workloads, monitor, and manage all user activity on these workspaces. They can also add the MLUser and MLBusinessUser roles to their assigned environment. This user also needs the account-level role of IAMViewer, in order to access the environment Manage Access page. To create or delete workspaces, this user also needs the EnvironmentAdmin role.
- MLUser: Grants a CDP user the ability to view Cloudera Machine Learning workspaces provisioned within a given CDP environment. MLUsers will also be able to run workloads on all the workspaces provisioned within this environment.
- MLBusinessUser: Grants permission to list Cloudera Machine Learning workspaces for a given CDP environment. MLBusinessUsers are able to only view applications deployed under the projects that they have been added to as a Business User.

Business Users and CML

A user is treated as a Business User inside of CML if they are granted the MLBusinessUser role on the Environment of the given ML Workspace. Inside of the Workspace, a Business User is able to access and view applications, but does not have privileges to access any other workloads in the Workspace.

Logging in as a Business User

When you log in as a Business User, the only page you see is the Applications page. The page shows any applications associated with any projects that you have been added to as a Collaborator, even though you do not have rights to access the other assets associated with those projects.

In order for applications to appear in your view, contact the Project Owner to add you as a Collaborator to the project. If you have not been added to any projects, or none of the projects that you have been added to have applications, the Applications page displays the message, You currently don’t have any applications.

Managing your Personal Account

You can edit personal account settings such as email, SSH keys and Hadoop credentials.

About this task

You can also access your personal account settings by clicking Account settings in the upper right-hand corner drop-down menu. This option will always take you to your personal settings page, irrespective of the context you are currently in.

Procedure

1. Sign in to Cloudera Machine Learning.
2. From the upper right drop-down menu, switch context to your personal account.
3. Click Settings.

   Profile

   You can modify your name, email, and bio on this page.

   Teams

   This page lists the teams you are a part of and the role assigned to you for each team.

   SSH Keys

   Your public SSH key resides here. SSH keys provide a useful way to access to external resources such as databases or remote Git repositories. For instructions, see *SSH Keys*.

Related Information

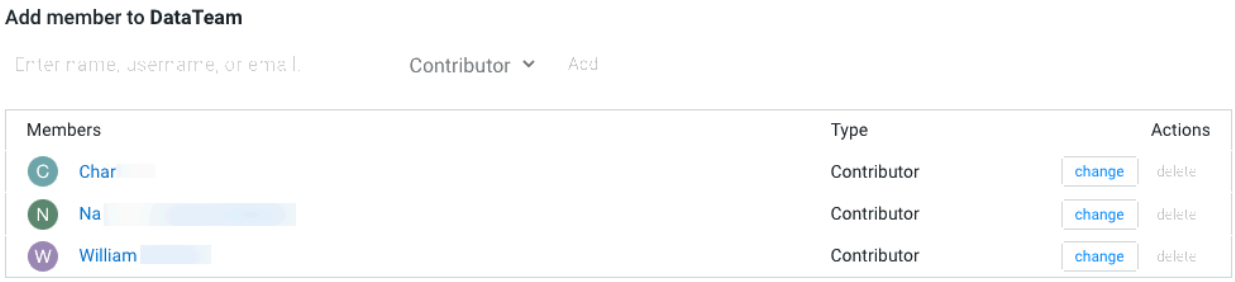
SSH Keys

Creating a Team

Users who work together on more than one project and want to facilitate collaboration can create a Team. Teams enable you to efficiently manage the users assigned to projects.

About this task

Team projects are owned by the team, rather than an individual user. Team administrators can add or remove members at any time, assigning each member different permissions.



Procedure

1. In Site Administration Teams , select New Team.
2. Enter the name of the team.
3. Select Local or Synced Team.

   CDP manages the member data of a Synced Team. The member data of a Local team is not managed by CDP.
4. If Synced Team is selected, choose a group in Select Synced Group.

5. Enter a Description, if needed.
6. Add or invite team members. Team members can have one of the following privilege levels:
   - Viewer - Read-only access to team projects. Cannot create new projects within the team but can be added to existing ones.
   - Operator - Read-only access to team projects. Additionally, Operators can start and stop existing jobs in the projects that they have access to.
   - Contributor - Write-level access to all team projects to all team projects with Team or Public visibility. Can create new projects within the team. They can also be added to existing team projects.
   - Admin - Has complete access to all team projects, can add new team members, and modify team account information.
7. Select Create Team.

Managing a Team Account

Team administrators can modify account information, add or invite new team members, and view/edit privileges of existing members.

Procedure

1. From the upper right drop-down menu, switch context to the team account.
2. Click Settings to open up the Account Settings dashboard.
3. Modify any of the following settings:

   Profile

   Modify the team description on this page.

   Members

   You can add new team members on this page, and modify privilege levels for existing members.

   SSH Keys

   The team's public SSH key resides here. Team SSH keys provide a useful way to give an entire team access to external resources such as databases. For instructions, see *SSH Keys*. Generally, team SSH keys should not be used to authenticate against Git repositories. Use your personal key instead.

Related Information

SSH Keys

Collaborating on Projects with Cloudera Machine Learning

This topic discusses all the collaboration strategies available to Cloudera Machine Learning users.

Project Collaborators

If you want to work closely with trusted colleagues on a particular project, you can add them to the project as collaborators. This is recommended for collaboration over projects created under your personal account. Anyone who belongs to your organization can be added as a project collaborator.

Project Visibility Levels: When you create a project in your personal context, Cloudera Machine Learning asks you to assign one of the following visibility levels to the project - Private or Public. Public projects on Cloudera Machine Learning grant read-level access to everyone with access to the Cloudera Machine Learning application. For Private projects, you must explicitly add someone as a project collaborator to grant them access.

Project Collaborator Access Levels: You can grant project collaborators the following levels of access: Viewer, Operator, Contributor, Admin

Note:

Collaborating Securely on Projects

Before adding project collaborators, you must remember that assigning the Contributor or Admin role to a project collaborator is the same as giving them write access to your data in CDH. This is because project contributors and project administrators have write access to all your project code (including any library code that you might not be actively inspecting). For example, a contributor/admin could modify project file(s) to insert code that deletes some data on the cluster. The next time you launch a session and run the same code, it will appear as though you deleted the data yourself.

Additionally, project collaborators also have access to all actively running sessions and jobs by default. This means that a malicious user can easily impersonate you by accessing one of your active sessions. Therefore, it is extremely important to restrict project access to trusted collaborators only. Note that site administrators can restrict this ability by allowing only session creators to run commands within their own active sessions.

For these reasons, Cloudera recommends using Git to collaborate securely on shared projects.

Teams

Users who work together on more than one project and want to facilitate collaboration can create a Team. Teams allow streamlined administration of projects. Team projects are owned by the team, rather than an individual user. Only users that are already part of the team can be added as collaborators to projects created within the team context. Team administrators can add or remove members at any time, assigning each member different access permissions.

Team Member Access Levels: You can grant team members the following levels of access: Viewer, Operator, Contributor, Admin.

ML Business User

The ML Business User role is for a user who only needs to view any applications that are created within Cloudera Machine Learning. This is the ideal role for an employee who is not part of the Data Science team and does not need higher-level access to workspaces and projects, but needs to access the output of a Data Science workflow. MLBusinessUser seats are available for purchase separately.

Forking Projects

You can fork another user's project by clicking Fork on the Project page. Forking creates a new project under your account that contains all the files, libraries, configurations, jobs, and dependencies between jobs from the original project.

Creating sample projects that other users can fork helps to bootstrap new projects and encourage common conventions.

Collaborating with Git

Cloudera Machine Learning provides seamless access to Git projects. Whether you are working independently, or as part of a team, you can leverage all of benefits of version control and collaboration with Git from within Cloudera Machine Learning. Teams that already use Git for collaboration can continue to do so. Each team member will need to create a separate Cloudera Machine Learning project from the central Git repository.

For anything but simple projects, Cloudera recommends using Git for version control. You should work on Cloudera Machine Learning the same way you would work locally, and for most data scientists and developers that means using Git.

Sharing Job and Session Console Outputs

This topic describes how to share the results of your research (that is, output from sessions and jobs) with teammates and project stakeholders.

Cloudera Machine Learning lets you easily share the results of your analysis with one click. Using rich visualizations and documentation comments, you can arrange your console log so that it is a readable record of your analysis and results. This log continues to be available even after the session stops. This method of sharing allows you to show colleagues and collaborators your progress without your having to spend time creating a report.

To share results from an interactive session, click Share at the top of the console page. From here you can generate a link that includes a secret token that gives access to that particular console output. For jobs results, you can either share a link to the latest job result or a particular job run. To share the latest job result, click the Latest Run link for a job on the Overview page. This link will always have the latest job results. To share a particular run, click on a job run in the job's History page and share the corresponding link.

You can share console outputs with one of the following sets of users.

- All anonymous users with the link - By default, Cloudera Machine Learning allows anonymous access to shared consoles. However, site administrators can disable anonymous sharing at any time.

  Once anonymous sharing has been disabled, all existing publicly shared console outputs will be updated to be viewable only by authenticated users.
- All authenticated users with the link - This means any user with a Cloudera Machine Learning account will have access to the shared console.
- Specific users and teams - Click Change to search for users and teams to give access to the shared console. You can also come back to the session and revoke access from a user or team the same way.

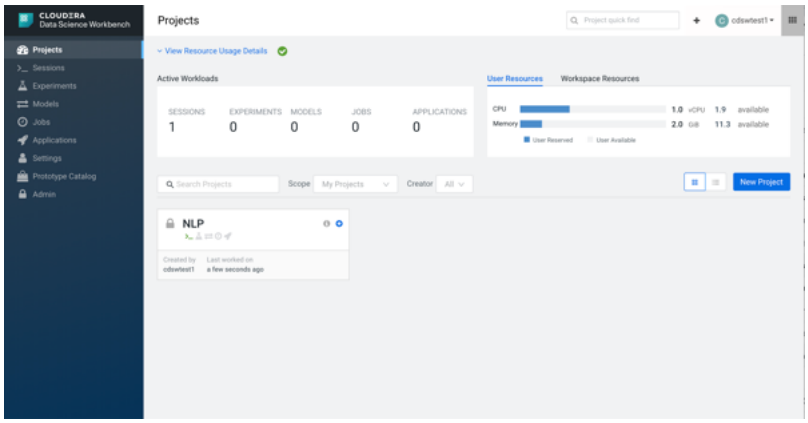
Sharing Data Visualizations

If you want to share a single data visualization rather than an entire console, you can embed it in another web page. Click the small circular 'link' button located to the left of most rich visualizations to view the HTML snippet that you can use to embed the visualization.

Projects in Cloudera Machine Learning

Projects form the heart of Cloudera Machine Learning. They hold all the code, configuration, and libraries needed to reproducibly run analyses. Each project is independent, ensuring users can work freely without interfering with one another or breaking existing workloads.

Access the Projects page by clicking Projects in the navigation panel. The Projects page gives you a quick summary of project information.



- Active Workloads - If there are active workloads running, this section describes the number of Sessions, Experiments, Models, Jobs, and Applications that are running.

- Resource Usage Details - A collapsible section that displays resource usage.
  - Active Workloads - If there are active workloads running, this section describes the number of Sessions, Experiments, Models, Jobs, and Applications that are running.
  - User Resources and Workspace Resources
    - Click on the User Resources tab to see the CPU and memory resource usage for the user. The maximum usage of the vCPU and GB is calculated based on whether or not you have a quota. If you have a quota, the maximum usage will be based on your quota. If you don't have a quota, the maximum usage will be what is available on the cluster. If you have a GPU, you'll also see the GPU usage.
    - Click on the Workspace Resources tab to see usage overall.
- Search Projects - Enter a term for keyword search across Project names.
- Scope - An additional filter only viewable by Administrators.
  - Selecting My Projects displays only the Projects that you have created or are a Collaborator of.
  - Selecting All Projects displays all Projects on the ML Workspace.
- Creator - An additional filter to only display Projects created by a specified user.
- Projects View Selector - A setting that enables you to display Projects in a summary card-based view or a detailed table-based view.

The following topics describe how to create and manage projects in Cloudera Machine Learning.

Creating a Project with Legacy Engine Variants

Projects create an independent working environment to hold your code, configuration, and libraries for your analysis. This topic describes how to create a project with Legacy Engine variants in Cloudera Machine Learning.

Procedure

1. Go to Cloudera Machine Learning and on the left sidebar, click Projects.
2. Click New Project.
3. If you are a member of a team, from the drop-down menu, select the Account under which you want to create this project. If there is only one account on the deployment, you will not see this option.
4. Enter a Project Name.
5. Select Project Visibility from one of the following options.
   - Private - Only project collaborators can view or edit the project.
   - Team - If the project is created under a team account, all members of the team can view the project. Only explicitly-added collaborators can edit the project.
   - Public - All authenticated users of Cloudera Machine Learning will be able to view the project. Collaborators will be able to edit the project.
6. Under Initial Setup, you can either create a blank project, or select one of the following sources for your project files.
   - Built-in Templates - Template projects contain example code that can help you get started with Cloudera Machine Learning. They are available in R, Python, PySpark, and Scala. Using a template project is not required, but it helps you start using Cloudera Machine Learning right away.
   - Custom Templates - Site administrators can add template projects that are customized for their organization's use-cases. For details, see *Custom Template Projects*.
   - Local - If you have an existing project on your local disk, use this option to upload compressed files or folders to Cloudera Machine Learning.
   - Git - If you already use Git for version control and collaboration, you can continue to do so with Cloudera Machine Learning. Specifying a Git URL will clone the project into Cloudera Machine Learning. If you use a Git SSH URL, a private SSH key associated with your CML user account is used to clone the repository. This is the recommended approach if you plan to contribute back to the cloned repository. However, you must add

the public SSH key from your personal Cloudera Machine Learning account to the remote Git hosting service before you can clone the project.

Select the correct format for the GitHub URL for the repository, and paste it into the text box.

In GitHub, the “Code” button in a repository shows you how to clone the repository.

If authentication is not required, use the format shown for Clone with HTTPS:

```
https://github.com/<example-org>/<example-project>.git
```

If you are cloning a Git repository that requires authentication, use this format for SSH:

```
git@github.com:<example-org>/<example-project>.git
```

With SSH, you need to first upload your public key to GitHub. Also, if you plan to perform GitHub operations such as `git push` or `git commit`, you should always use GitHub over SSH.

7. Click Create Project. After the project is created, you can see your project files and the list of jobs defined in your project.

   Note that as part of the project filesystem, Cloudera Machine Learning also creates the following .gitignore file.

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

8. (Optional) To work with team members on a project, add them as collaborators to the project.

Related Information

Custom Template Projects

Adding a new SSH key to your GitHub account

Configuring Project-level Runtimes

If you've specified project-level Runtimes, you can view your chosen Runtime configuration by clicking Project Settings Runtime/Engine . Your chosen Runtimes are listed under Available Runtimes.

- If the Available Runtimes table is empty, users can select from all Runtimes available in the deployment to start new sessions or workloads.
- The filtering options in Project Settings only affect the user interface. The filtering options do not apply when projects are accessed using API v2.
- You can remove an available Runtime by clicking the corresponding "x" in the right most column. Runtimes can only be removed if there are no active workloads using them.
- You can add additional Runtimes by clicking Add Runtime and chosing additional Runtimes.
- If there is a newer version of a Runtime, a warning icon displays next to the appropriate Runtimes. You can apply the latest version by clicking Add Latest. The older Runtime version is not removed from the table. However, you can remove it by clicking the x" in the right most column.
- The Available Runtimes table shows related counters for non-interactive workloads.

Adding Project Collaborators

This topic shows you how to invite colleagues to collaborate on a project.

About this task

For a project created under your personal account, anyone who belongs to your organization can be added as a collaborator. For a project created under a team account, you can only add collaborators that already belong to the team. If you want to work on a project that requires collaborators from different teams, create a new team with the

required members, and then create a project under that account. If your project was created from a Git repository, each collaborator must create the project from the same central Git repository.

You can grant project collaborators one of three levels of access:

- Viewer - Read-only access to code, data, and results.
- Operator - Read-only access to code, data, and results. Additionally, Operators can start and stop existing jobs in the projects that they have access to.
- Contributor - Can view, edit, create, and delete files and environmental variables, run sessions/experiments/jobs/models and run code in running jobs. Additionally, Contributors can set the default engine for the project.
- Admin - Has complete access to all aspects of the project. This includes the ability to add new collaborators, and delete the entire project.

Note:

Collaborating Securely on Projects

Before adding project collaborators, you must remember that assigning the Contributor or Admin role to a project collaborator is the same as giving them write access to your data in CDH. This is because project contributors and project administrators have write access to all your project code (including any library code that you might not be actively inspecting). For example, a contributor/admin could modify project file(s) to insert code that deletes some data on the CDH cluster. The next time you launch a session and run the same code, it will appear as though you deleted the data yourself.

Additionally, project collaborators also have access to all actively running sessions and jobs. This means that a malicious user can easily impersonate you by accessing one of your active sessions. Therefore, it is extremely important to restrict project access to trusted collaborators only. Note that site administrators can restrict this ability by allowing only session creators to run commands within their own active sessions.

For these reasons, Cloudera recommends using Git to collaborate securely on shared projects. This will also help avoid file modification conflicts when your team is working on more elaborate projects.

Procedure

1. In Cloudera Machine Learning, navigate to the project overview page.
2. Click Team to open the Collaborators page.
3. Search for collaborators by either name or email address and click Add.

Modifying Project Settings

Project contributors and administrators can modify aspects of the project environment such as the engine used to launch sessions, the environment variables, and to create SSH tunnels to access external resources.

Procedure

1. Switch context to the account where the project was created.
2. Click Projects.
3. From the list of projects, select the one to modify.

4. Click Project Settings to open the Project Settings dashboard.

   Options

   Modify the project name and its privacy settings on this page.

   Engine

   Cloudera Machine Learning ensures that your code is always run with the specific engine version you selected. You can also select the engine version and add third-party editors here.

   Advanced

   - Environment Variables - If there are any environmental variables that should be injected into all the engines running this project, you can add them to this page. For more details, see *Engine Environment Variables*.
   - Shared Memory Limit - You can specify additional shared memory available to sessions running with the project.

     Note: You can specify additional shared memory available to sessions running with the project. The maximum size of this volume is the half of your physical RAM in the node, not including memory used for swap.

   Tunnels

   In some environments, external databases and data sources reside behind restrictive firewalls. Cloudera Machine Learning provides a convenient way to connect to such resources using your SSH key. For instructions, see *SSH Keys*.

   Delete Project

   This page can only be accessed by project administrators. Remember that deleting a project is irreversible. All files, data, sessions, and jobs are removed.

Related Information

Managing Engines

Engine Environment Variables

SSH Keys

Managing Project Files

Cloudera Machine Learning allows you to move, rename, copy, and delete files within the scope of the project where they live. You can also upload new files to a project, or download project files. For use cases beyond simple projects, Cloudera strongly recommends using *Git for Collaboration* to manage your projects using version control.

Procedure

1. Switch context to the account where the project was created.
2. Click Projects.
3. From the list of projects, click on the project you want to modify. This will take you to the project overview.

4. Click Files.

   Upload Files to a Project

   Files can only be uploaded within the scope of a single project. Therefore, to access a script or data file from multiple projects, you will need to manually upload it to all the relevant projects.

   Click Upload. Select Files or Folder from the dropdown, and choose the files or folder you want to upload from your local filesystem.

   In addition to uploading files or a folder, you can upload a .tar file of multiple files and folders. After you select and upload the .tar file, you can use a terminal session to extract the contents:

   a. On the project overview page, click Open Workbench and select a running session or create a new one.
   b. Click Terminal access.
   c. In the terminal window, extract the contents of the .tar file:

      ```
      tar -xvf <file_name>.tar.gz
      ```

      The extracted files are now available for the project.

   Download Project Files

   Click Download to download the entire project in a .zip file. To download only a specific file, select the checkbox next to the file(s) to be download and click Download.

5. You can also use the checkboxes to Move, Rename, or Delete files within the scope of this project.

Related Information

Git for Collaboration

Custom Template Projects

Site administrators can add template projects that have been customized for their organization's use-cases. These custom project templates can be added in the form of a Git repository.

Required Role: See User Role Authorization.

To add a new template project, go to Admin Settings . Under the Project Templates section, provide a template name, the URL to the project's Git repository, and click Add.

The added templates will become available in the Template tab on the Create Project page. Site administrators can add, edit, or delete custom templates, but not the built-in ones. However, individual built-in templates can be disabled using a checkbox in the Project Templates table at Admin Settings .

Deleting a Project

This topic demonstrates how to delete a project.

About this task

Important: Deleting a project is an irreversible action. All files, data, and history related to the project will be lost. This includes any jobs, sessions or models you created within the project.

Procedure

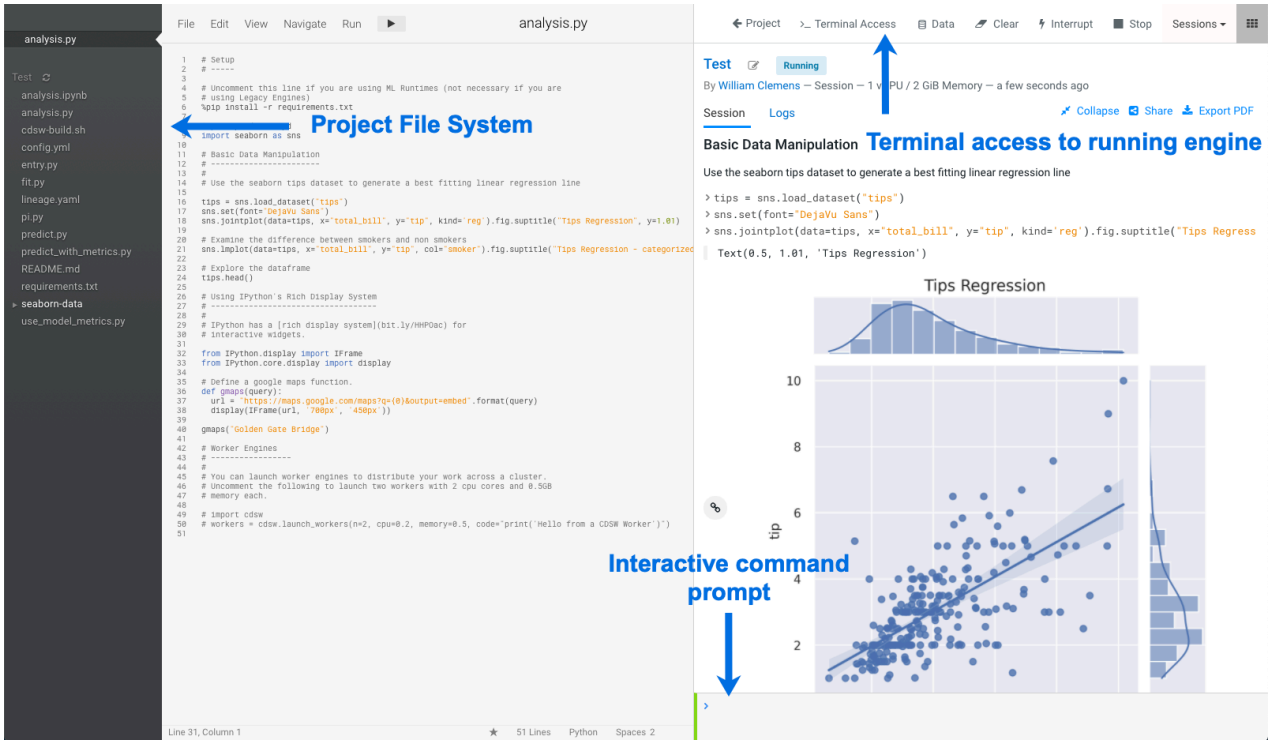
1. Go to the project Overview page.
2. On the left sidebar, click Settings.
3. Go to the Delete Project.
4. Click Delete Project and click OK to confirm.

Native Workbench Console and Editor

The workbench console provides an interactive environment tailored for data science, supporting R, Python and Scala. It currently supports R, Python, and Scala engines. You can use these engines in isolation, as you would on your laptop, or connect to your CDH cluster.

The workbench UI includes four primary components:

- An editor where you can edit your scripts.
- A console where you can track the results of your analysis.
- A command prompt where you can enter commands interactively.
- A terminal where you can use a Bash shell.



Typically, you would use the following steps to run a project in the workbench:

Related Information

Managing Engines

Launch a Session

Sessions allow you to perform actions such as run R or Python code. They also provide access to an interactive command prompt and terminal. This topic demonstrates how to launch a new session.

Procedure

1. Navigate to your project's Overview page.
2. Click New Session.

3. Check the settings for your session:

   You see the following settings:

   Editor

   Selects the Editor; currently only Workbench is supported and therefore the selector is static.

   Kernel

   Selects the Kernel. Initially only Python Runtimes are supported.

   Engine Image

   Displays the Advanced tab in Project Settings and allows you to set environment variables and the shared memory limit.

4. You can modify the engine image used by this session:
   a) By Engine Image, click Configure.

      Cloudera Machine Learning displays the Project Settings page.
   b) Select the Runtime/Engine tab.
   c) Next to Default Engine, select ML Runtime or Legacy Engine.
   d) Click Save Engine.
5. Specify your Resource Profile.

   This attribute will define how many vCPUs and how much memory will be reserved to run the workload (for example, session including the runtime itself). The minimum configuration is 1vCPU and 2 GB memory.
6. Click Start Session.
   The command prompt at the bottom right of your browser window will turn green when the engine is ready. Sessions typically take between 10 and 20 seconds to start.

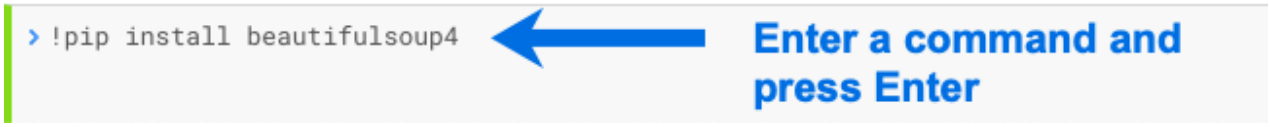
Run Code

This topic shows you how to enter and run code in the interactive Workbench command prompt or the editor after you launch a session.

The editor is best for code you want to keep, while the command prompt is best for quick interactive exploration.

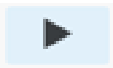
Command Prompt - The command prompt functions largely like any other. Enter a command and press Enter to run it. If you want to enter more than one line of code, use Shift+Enter to move to the next line. The output of your code, including plots, appears in the console.

```
> ls
analysis.ipynb  entry.py      predict.py               seaborn-data/
analysis.py     fit.py        predict_with_metrics.py  use_model_metrics.py
cdsw-build.sh*  lineage.yaml  README.md
config.yml      pi.py         requirements.txt
```



If you created your project from a template, you should see project files in the editor. You can open a file in the editor by clicking the file name in the file navigation bar on the left.

Editor - To run code from the editor:

1. Select a script from the project files on the left sidebar.
2. To run the whole script click  on the top navigation bar, or, highlight the code you want to run and press Ctrl+Enter (Windows/Linux) or cmd+Enter (macOS).

When doing real analysis, writing and executing your code from the editor rather than the command prompt makes it easy to iteratively develop your code and save it along the way.

If you require more space for your editor, you can collapse the file list by double-clicking between the file list pane and the editor pane. You can hide the editor using editor's View menu.

Code Autocomplete

The Python and R kernels include support for automatic code completion, both in the editor and the command prompt. Use single tab to display suggestions and double tab for autocomplete.

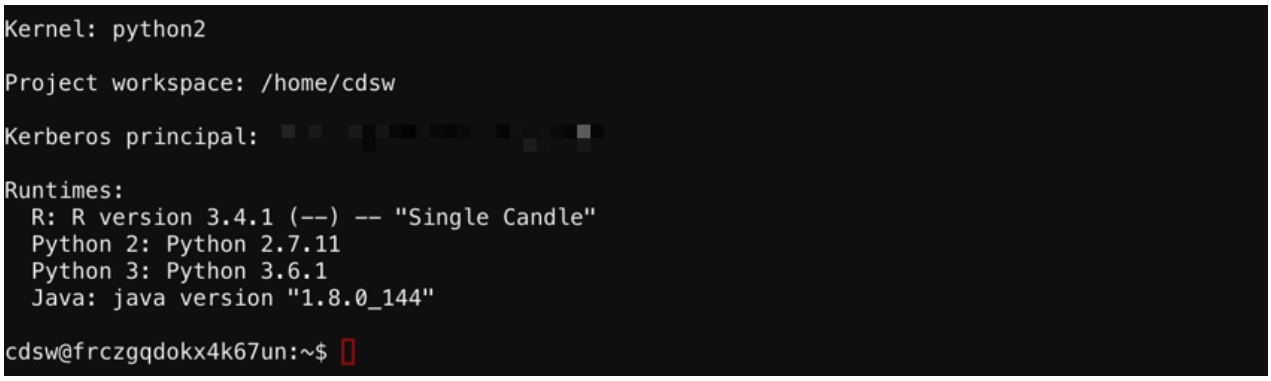
Project Code Files

All project files are stored to persistent storage within the respective project directory at /var/lib/cdsw/current/projects. They can be accessed within the project just as you would in a typical directory structure. For example, you can import functions from one file to another within the same project.

Access the Terminal

Cloudera Machine Learning provides full terminal access to running engines from the web console. This topic show you how to access the Terminal from a running Workbench session.

You can use the terminal to move files around, run Git commands, access the YARN and Hadoop CLIs, or install libraries that cannot be installed directly from the engine. To access the Terminal from a running session, click Terminal Access above the session log pane.



The terminal's default working directory is /home/cdsw, which is where all your project files are stored. Any modifications you make to this folder will persist across runs, while modifications to other folders are discarded.

If you are using Kerberos authentication, you can run klist to see your Kerberos principal. If you run hdfs dfs -ls you will see the files stored in your HDFS home directory.

Note that the terminal does not provide root or sudo access to the container. To install packages that require root access, see *Customized Engine Images*.

Related Information

Customized Engine Images

Stop a Session

This topic demonstrates how to stop a session to free up resources for other users when you are finished.

When you are done with the session, click Stop in the menu bar above the console, or use code to exit by typing the following command:

R

```
quit()
```

Python

```
exit
```

Scala

```
quit()
```

Sessions automatically stop after an hour of inactivity.

Workbench editor file types

The default workbench editor supports the following file types:

- Text
- CSS
- HTML
- JavaScript
- JSON
- PHP
- Scala
- C++
- C#
- CLike
- Java
- CoffeeScript
- R
- Julia
- Ruby
- Clojure
- Perl
- Python
- SASS
- Lua
- SQL
- Diff
- Markdown
- YAML
- Haxe

Third-Party Editors

In addition to the built-in Cloudera Machine Learning editor, you can configure Cloudera Machine Learning to work with third-party, browser-based IDEs such as Jupyter and also certain local IDEs that run on your machine, such as PyCharm.

Note: Custom editors run inside CML sessions. If the CML session is stopped, this may cause unexpected behavior in the editor UI and, in some cases, may result in data loss. You should, therefore, use the custom editor's UI to shut the editor down first. This will automatically end the CML session too.

In JupyterLab you do that by clicking "Shut Down" in the JupyterLab "File" menu." This applies to both engines and Runtimes, and all versions of CML.

When you bring your own editor, you still get many of the benefits Cloudera Machine Learning behind an editor interface you are familiar with:

- Dependency management that lets you share code with confidence
- CDH client configurations
- Automatic Kerberos authentication through Cloudera Machine Learning
- Reuse code in other Cloudera Machine Learning features such as experiments and jobs
- Collaboration features such as teams
- Compliance with IT rules for where compute, data, and/or code must reside. For example, compute occurs within the Cloudera Machine Learning deployment, not the local machine. Browser IDEs run within a Cloudera Machine Learning session and follow all the same compliance rules. Local IDEs, on the other hand, can bring data or code to a user's machine. Therefore, Site Administrators can opt to disable local IDEs to balance user productivity with compliance concerns.

In the Cloudera Machine Learning documentation, browser-based IDEs like Jupyter will be referred to as "browser IDEs". IDEs such as PyCharm that run on your machine outside of your browser will be referred to as "local IDEs" because they run on your local machine. You can use the browser or local IDE of your choice to edit and run code interactively.

Note that you can only edit and run code interactively with the IDEs. Tasks such as creating a project or deploying a model require the Cloudera Machine Learning web UI and cannot be completed through an editor.

Modes of Configuring Third-Party Editors

The configuration for an IDE depends on which type of editor you want to use.

In addition to the native Cloudera Machine Learning editor, you can configure Cloudera Machine Learning to work with third-party, browser-based IDEs, such as Jupyter, and also certain local IDEs that run on your machine, such as PyCharm.

Workbench editor

The Workbench editor is the built-in editor for Cloudera Machine Learning. No additional configuration is required to use it. When you launch a session, select the Workbench editor.

Third-party, browser-based IDEs

Browser IDEs are editors such as Jupyter or RStudio. When you use a browser IDE, it runs within a session and allows you to edit and run code interactively. Changes that you make in the editor are propagated to the Cloudera Machine Learning project. Base Engine Image v8 and higher ships with Jupyter preconfigured as a browser IDE. You can select it when you start a session or add a different browser IDE. For more information, see *Configure a Browser IDE as an Editor*.

Keep the following in mind when using browser IDEs:

- Engine Version Requirements
  - Browser-based IDEs require Base Engine Image v8 or higher.
- When you are finished using a browser IDE, you must exit the IDE properly, including saving your work if necessary. Do not just stop the Cloudera Machine Learning session. Doing so will cause you to lose your session state. For example, if you want RStudio to save your state, including variables, to ~/.RData, exit the RStudio workspace using the power button in the top right of the RStudio UI.
- Depending on the behavior of the browser IDE, multiple users within a project may overwrite each other's state. For example, RStudio state is persisted in /home/cdsw/.RData that is shared by all users within a project.
- Browser IDEs do not adhere to the timeout set in IDLE_MAXIMUM_MINUTES. Instead, they use the timeout set in SESSION_MAXIMUM_MINUTES, which is 7 days by default. Cloudera recommends that users stop their session manually after using a browser-based editor. Running sessions continue to consume resources and may impact other users.
- Logs for browser IDEs are available on the Logs tab of the session window. This includes information that the IDE may generate, such as error messages, in addition to any Cloudera Machine Learning logs.

Local IDE Editors on your machine that can use SSH-based remote editing

These editors, referred to as Local IDEs in the documentation, are editors such as PyCharm that run on your local machine. They connect to Cloudera Machine Learning with an SSH endpoint and allow you to edit and run code interactively. You must manually configure some sort of file sync and ignore list between your local machine and Cloudera Machine Learning. You can use functionality within the local IDE, such as PyCharm's sync, or external tools that can sync via the SSH endpoint, such as Mutagen.

Keep the following in mind before setting up local IDEs:

- Local IDEs do not require a specific engine image, but Cloudera always recommends you use the latest engine image.
- Site Administrators should work with IT to determine the data access policies for your organization. For example, your data policy may not allow users to sync certain files to their machines from Cloudera Machine Learning. Verify that users understand the requirements and adhere to them when configuring their file sync behavior.
- Users should ensure that any IDEs that the IDEs they want to use support SSH. For example, VS Code supports "remote development over SSH," and PyCharm supports using a "remote interpreter over SSH."

Related Information

Configure a Browser IDE as an Editor
Configure a Local IDE using an SSH Gateway

Configure a Browser IDE as an Editor

When you use a browser IDE, changes that you make in the editor are propagated to the Cloudera Machine Learning project.

About this task

For example, if you create a new .py file or modify an existing one with the third-party editor, the changes are propagated to Cloudera Machine Learning. When you run the code from the IDE, execution is pushed from the IDE to Cloudera Machine Learning.

Base Engine Image v8 and higher for Cloudera Machine Learning comes preconfigured with Jupyter, and any browser IDEs you want to add must be added to Base Engine Image v8 or higher. Jupyter can be selected in place of the built-in Workbench editor when you launch a session, and no additional configuration is required. You can configure additional IDEs to be available from the dropdown.

You have two configuration options:

- Project Level: You can configure an editor at the project level so that any session launched within that project can use the editor configured. Other projects across the deployment will not be able to use any editors configured in such a manner. For steps, see *Configure a Browser IDE at the Project Level*.
- Engine Level: You can create a custom engine configured with the editor so that any project across the deployment that uses this custom engine can also use the editor configured. This might be the only option in case of certain browser IDEs (such as RStudio) that require root permission to install and therefore cannot be directly installed within the project. For steps, see Configure a Browser IDE at the Engine Level.

Cloudera recommends you first test the browser IDE you intend to install in a session before you install it to the project or build a custom engine with it. For steps, see *Test a Browser IDE in a Session Before Installation*.

Test a Browser IDE in a Session Before Installation

This process can be used to ensure that a browser IDE works as expected before you install it to a project or to a customized engine image. This process is not meant for browser IDEs that require root permission to install, such as RStudio.

About this task

These steps are only required if you want to use an editor that does not come preinstalled as part of the default engine image. Perform the following steps to configure an editor for your session:

Procedure

1. Ensure that your browser accepts pop-up windows and cookies from Cloudera Machine Learning web UI.
2. Open the Cloudera Machine Learning web UI.
3. Go to your project and launch a session with the kernel of your choice and the Workbench editor. Alternatively, open an existing session.
4. In the interactive command prompt or terminal for the session, install the editor you want to use. See the documentation for your editor for specific instructions.

   For example:

   Jupyter Lab

   Python 3

   The following example command installs Jupyter Lab for Python 3:

   ```
   !pip3 install jupyterlab
   ```

5. After the installation completes, enter the command to start the server for the notebook on the port specified in the CDSW_APP_PORT environment variable on IP address 127.0.0.1.

   For example, the following command starts the server for Jupyter Lab on the port specified in the CDSW_APP_PORT environment variable:

   ```
   !/home/cdsw/.local/bin/jupyter-lab --no-browser --ip=127.0.0.1 --port=${
   CDSW_APP_PORT} --NotebookApp.token= --NotebookApp.allow_remote_access=True
    --log-level=ERROR
   ```

6. Click on the grid icon in the top right.
   You should see the editor in the drop-down menu. If you select the editor, it opens in a new browser tab.

Configure a Browser IDE at the Project Level

The following steps are only required if you want to use an editor that is not included in the default engine image that ships with Cloudera Machine Learning.

Before you begin

Before you start, verify that you have installed the IDE of your choice to the project. For information about how to install additional packages to a project, see *Installing Additional Packages*.

About this task

Perform the following steps to add an editor to a project:

Procedure

1. Open the Cloudera Machine Learning web UI.
2. Go to the project you want to configure an editor for.
3. Go to Settings Editors and click New Editor.

4. Complete the fields:
   - Name: Provide a name for the editor. This is the name that appears in the dropdown menu for Editors when you start a new session.
   - Command: Enter the command to start the server for the editor on the Cloudera Machine Learning public port specified in the CDSW_APP_PORT environment variable (default 8081).

     For example, the following command starts Jupyter Lab on the port specified by the CDSW_APP_PORT environment variable:

     ```
     /home/cdsw/.local/bin/jupyter-lab --no-browser --ip=127.0.0.1 --port=${C
     DSW_APP_PORT} --NotebookApp.token= --NotebookApp.allow_remote_access=Tru
     e --log-level=ERROR
     ```

     This is the same command you used to start the IDE to test it in a session.
5. Save the changes.
   When a user starts a new session, the editor you added is available in the list of editors. Browsers must be configured to accept cookies and allow pop-up windows from the Cloudera Machine Learning web UI.

Related Information

Installing Additional Packages

Configure a Browser IDE at the Legacy Engine Level

You can make a browser IDE available to any project within a Cloudera Machine Learning deployment by creating a customized legacy engine image, installing the editor to it, and adding it to the trusted list for a project. Additionally, browser IDEs that require root permission to install, such as RStudio, can only be used as part of a customized legacy engine image.

About this task

When a user launches a session, they can select the customized legacy engine with the editors available. The following steps describe how to make a customized legacy engine image for RStudio:

Procedure

1. Create a Dockerfile for the new custom image. Note that the Base Engine Image uses Ubuntu, and you must use Base Engine Image v9 or higher.

   The following sample Dockerfile is for RStudio:

   ```
   #Dockerfile
   FROM docker.repository.cloudera.com/cdsw/engine:9-cml1.1

   WORKDIR /tmp

   #Delete the Cloudera repository that is inaccessible because of the payw
   all
   RUN rm /etc/apt/sources.list.d/*

   #The RUN commands that install an editor
   #For example: RUN apt-get install myeditor

   RUN apt-get update && apt-get dist-upgrade -y && \
       apt-get install -y --no-install-recommends \
       libapparmor1 \
       libclang-dev \
       lsb-release \
       psmisc \
       sudo
   ```

```
#The command that follows RUN is the same command you used to install the
 IDE to test it in a the session.
RUN wget https://download2.rstudio.org/server/trusty/amd64/rstudio-server-
1.2.1335-amd64.deb && \
    dpkg -i rstudio-server-1.2.1335-amd64.deb

COPY rserver.conf /etc/rstudio/rserver.conf

COPY rstudio-cdsw /usr/local/bin/rstudio-cdsw

RUN chmod +x /usr/local/bin/rstudio-cdsw
```

2. Create rserver.conf:

```
# Must match CDSW_APP_PORT
www-port=8090
server-app-armor-enabled=0
server-daemonize=0
www-address=127.0.0.1
auth-none=1
auth-validate-users=0
```

Make sure that the www-port property matches the port set in the CDSW_APP_PORT environment variable (default 8090).

3. Create rstudio-cdsw:

```
#!/bin/bash
# This saves RStudio's user runtime information to /tmp, which ensures
several
# RStudio sessions can run in the same project simultaneously
mkdir -p /tmp/rstudio/sessions/active
mkdir -p /home/cdsw/.rstudio/sessions
if [ -d /home/cdsw/.rstudio/sessions/active ]; then rm -rf /home/cdsw/.rst
udio/sessions/active; fi
ln -s /tmp/rstudio/sessions/active /home/cdsw/.rstudio/sessions/active
# This ensures RStudio picks up the environment. This may not be necess
ary if
# you are installing RStudio Professional. See
# https://docs.rstudio.com/ide/server-pro/r-sessions.html#customizing-sess
ion-launches.
# SPARK_DIST_CLASSPATH is treated as a special case to workaround a bug in
 R
# with very long environment variables.
env | grep -v ^SPARK_DIST_CLASSPATH >> /usr/local/lib/R/etc/Renviron.site
echo "Sys.setenv(\"SPARK_DIST_CLASSPATH\"=\"${SPARK_DIST_CLASSPATH}\")"
 >> /usr/local/lib/R/etc/Rprofile.site

# Now start RStudio
/usr/sbin/rstudio-server start
```

4. Build the Dockerfile:

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

If you want to build your image on a 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
```

5. Distribute the image:
   - Push the image to a public registry such as DockerHub.

     For instructions, refer the Docker documentation.
   - 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>
     ```
6. Add the image to the trusted list in Cloudera Machine Learning:
   a) Log in to the Cloudera Machine Learning web UI as a site administrator.
   b) Click Admin Engines .
   c) Add <company-registry>/<user-name>/<image-name>:<tag> to the list of trusted engine images.
7. Add the new legacy engine to the trusted list for a project:
   a) Go to the project Settings page.
   b) Click Engines.
   c) Select the new customized legacy engine from the dropdown list of available Docker images. Sessions and jobs you run in your project will have access to this engine.
8. Configure RStudio for the project. When this is done, you will be able to select RStudio from the dropdown list of editors on the Launch New Sesssion page.
   a) Go to Settings > Editors and click New Editor.
   b) Complete the fields:
      - Name: Provide a name for the editor. For example, RStudio. This is the name that appears in the dropdown menu for Editors when you start a new session.
      - Command: Enter the command to start the server for the editor.

        For example, the following command will start RStudio:

        ```
        /usr/local/bin/rstudio-cdsw
        ```
   c) Save the changes.

Related Information

Docker push

Limitations

Configure a Local IDE using an SSH Gateway

The specifics for how to configure a local IDE to work with Cloudera Machine Learning are dependent on the local IDE you want to use.

Cloudera Machine Learning relies on the SSH functionality of the editors to connect to the SSH endpoint on your local machine created with the cdswctl client. Users establish an SSH endpoint on their machine with the cdswctl client. This endpoint acts as the bridge that connects the editor on your machine and the Cloudera Machine Learning deployment.

The following steps are a high-level description of the steps a user must complete:

1. Establish an SSH endpoint with the CML CLI client. See Initialize an SSH Endpoint.
2. Configure the local IDE to use Cloudera Machine Learning as the remote interpreter.
3. Optionally, sync files with tools (like mutagen, SSHFS, or the functionality built into your IDE) from Cloudera Machine Learning to your local machine. Ensure that you adhere to IT policies.

4. Edit the code in the local IDE and run the code interactively on Cloudera Machine Learning.
5. Sync the files you edited locally to Cloudera Machine Learning.
6. Use the Cloudera Machine Learning web UI to perform actions such as deploying a model that uses the code you edited.

You can see an end-to-end example for PyCharm configuration in the *CML Editors Pycharm*.

Configure PyCharm as a Local IDE

Cloudera Machine Learning supports using editors on your machine that allow remote execution and/or file sync over SSH, such as PyCharm.

About this task

This topic describes the tasks you need to perform to configure Cloudera Machine Learning to act as a remote SSH interpreter for PyCharm. Once finished, you can use PyCharm to edit and sync the changes to Cloudera Machine Learning. To perform actions such as deploying a model, use the Cloudera Machine Learning web UI.

Note: These instructions were written for the Professional Edition of PyCharm Version 2019.1. See the documentation for your version of PyCharm for specific instructions.

Before you begin, ensure that the following prerequisites are met:

- You have an edition of PyCharm that supports SSH, such as the Professional Edition.
- You have an SSH public/private key pair for your local machine.
- You have Contributor permissions for an existing Cloudera Machine Learning project. Alternatively, create a new project you have access to.

Add Cloudera Machine Learning as an Interpreter for PyCharm

In PyCharm, you can configure an SSH interpreter. Cloudera Machine Learning uses this method to connect to PyCharm and act as its interpreter.

About this task

Before you begin, ensure that the SSH endpoint for Cloudera Machine Learning is running on your local machine. These instructions were written for the Professional Edition of PyCharm Version 2019.1 and are meant as a starting point. If additional information is required, see the documentation for your version of PyCharm for specific instructions.

Procedure

1. Open PyCharm.
2. Create a new project.
3. Expand Project Interpreter and select Existing interpreter.
4. Click on ... and select SSH Interpreter
5. Select New server configuration and complete the fields:
   - Host: localhost
   - Port: 2222
   - Username: cdsw
6. Select Key pair and complete the fields using the RSA private key that corresponds to the public key you added to the Remote Editing tab in the Cloudera Machine Learning web UI..

   For macOS users, you must add your RSA private key to your keychain. In a terminal window, run the following command:

   ```
   ssh-add -K <path to your prviate key>/<private_key>
   ```

7. Complete the wizard. Based on the Python version you want to use, enter one of the following parameters:
   - /usr/local/bin/python2
   - /usr/local/bin/python3

   You are returned to the New Project window. Existing interpreter is selected, and you should see the connection to Cloudera Machine Learning in the Interpreter field.
8. In the Remote project location field, specify the following directory:

   ```
   /home/cdsw
   ```
9. Create the project.

Configure PyCharm to use Cloudera Machine Learning as the Remote Console

Procedure

1. In your project, go to Settings and search for Project Interpreter.

   Depending on your operating system, Settings may be called Preferences.
2. Click the gear icon and select Show All.
3. Select the Remote Python editor that you added, which is connected to the Cloudera Machine Learning deployment.
4. Add the following interpreter path by clicking on the folder icon:

   ```
   /usr/local/bin/python2.7/site-packages
   ```

(Optional) Configure the Sync Between Cloudera Machine Learning and PyCharm

Configuring what files PyCharm ignores can help you adhere to IT policies.

About this task

Before you configure syncing behavior between the remote editor and Cloudera Machine Learning, ensure that you understand the policies set forth by IT and the Site Administrator. For example, a policy might require that data remains within the Cloudera Machine Learning deployment but allow you to download and edit code.

Procedure

1. In your project, go to Settings and search for Project Interpreter.

   Depending on your operating system, Settings may be called Preferences.
2. Search for Deployment.
3. On the Connection tab, add the following path to the Root path field:

   ```
   /home/cdsw
   ```
4. Optionally, add a Deployment path on the Mappings tab if the code for your Cloudera Machine Learning project lives in a subdirectory of the root path.
5. Expand Deployment in the left navigation and go to Options Upload changed files automatically to the default server and set the behavior to adhere to the policies set forth by IT and the Site Administrator.

   Cloudera recommends setting the behavior to Automatic upload because the data remains on the cluster while your changes get uploaded.
6. Sync for the project file(s) to your machine and begin editing.

Configure VS Code as a Local IDE

About this task

Cloudera Machine Learning supports using local IDEs on your machine that allow remote execution and/or file sync over SSH, such as VS Code. This topic describes the tasks you need to perform to configure Cloudera Machine Learning to act as a remote SSH interpreter for VS Code. Once finished, you can use VS Code to edit and sync the changes to Cloudera Machine Learning. To perform actions such as deploying a model, use the Cloudera Machine Learning web UI.

Before you begin, ensure that the following prerequisites are met:

- You have an edition of VS Code that supports SSH.
- You have an SSH public/private key pair for your local machine that is compatible with VS Code.
- You have Contributor permissions for an existing Cloudera Machine Learning project. Alternatively, create a new project you have access to.

Download cdswctl and Add an SSH Key

The first step to configure VS Code as a local IDE is to download cdswctl and add an SSH key.

Procedure

1. Open the Cloudera Machine Learning web UI and go to Settings Remote Editing for your user account.
2. Download cdswctl client for your operating system.
3. In the terminal, run cat ~/.ssh/id_rsa.pub. If you used a different filename above when generating the key, use that filename instead. This command prints the key as a string.
4. Copy the key. It should resemble the following: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCha2J5 mW3i3BgtZ25/FOsxywpLVkx1RgmZunI
5. In SSH public keys for session access, paste the key.

What to do next

Cloudera Machine Learning uses the SSH public key to authenticate your CLI client session, including the SSH endpoint connection to the Cloudera Machine Learning deployment. Any SSH endpoints that are running when you add an SSH public key must also be restarted.

Initialize an SSH Connection to Cloudera Machine Learning for VS Code

The following task describes how to establish an SSH endpoint for Cloudera Machine Learning. Creating an SSH endpoint is the first step to configuring a remote editor for Cloudera Machine Learning.

Procedure

1. Log in to Cloudera Machine Learning with the CLI client.

   ```
   cdswctl login -n <username> -u http(s)://cdsw.your_domain.com
   ```

   For example, the following command logs the user sample_user into the https://cdsw.*your_domain*.com deployment:

   ```
   cdswctl login -n sample_user -u https://cdsw.your_domain.com
   ```

2. Create a local SSH endpoint to Cloudera Machine Learning.

   Run the following command:

   ```
   cdswctl ssh-endpoint -p <username>/<project_name> [-c <CPU_cores>] [-m <
   memory_in_GB>] [-g <number_of_GPUs>] [-r <runtime ID> ]
   ```

   If the project is configured to use ML runtimes, the -r parameter must be specified, otherwise it must be omitted. To retrieve the Runtime ID, use the following command:

   cdswctl runtimes list

   See *Using ML runtimes with cdswctl* documentation page for more information.

   The command uses the following defaults for optional parameters:

   - CPU cores: 1
   - Memory: 1 GB
   - GPUs: 0

   For example, the following command starts a session for the logged-in user sample_user under the customerchurn project with .5 cores, .75 GB of memory, 0 GPUs, and the Python3 kernel:

   ```
   cdswctl ssh-endpoint -p customerchurn -c 0.5 -m 0.75
   ```

   To create an SSH endpoint in a project owned by another user or a team, for example finance, prepend the username to the project and separate them with a forward slash:

   ```
   cdswctl ssh-endpoint -p finance/customerchurn -c 0.5 -m 0.75
   ```

   This command creates session in the project customerchurn that belongs to the team finance.

   Information for the SSH endpoint appears in the output:

   ```
   ...
   You can SSH to it using
       ssh -p <some_port> cdsw@localhost
   ...
   ```

3. Open a new command prompt and run the outputted command from the previous step:

   ```
   ssh -p <some_port> cdsw@localhost
   ```

   For example:

   ```
   ssh -p 7847 cdsw@localhost
   ```

   You will be prompted for the passphrase for the SSH key you entered in the Cloudera Machine Learning web UI.

   The public key could be rejected when the new ssh key pair is generated with a special name such as id_rsa_syste st. If the public key is rejected, you must add the following information to the ~/.ssh/config file:

   ```
   Host *
             AddKeysToAgent yes
             StrictHostKeyChecking no
             IdentityFile ~/.ssh/id_rsa_cdswctl
   ```

   Once you are connected to the endpoint, you are logged in as the cdsw user and can perform actions as though you are accessing the terminal through the Cloudera Machine Learning web UI.

4. Test the connection.

   If you run ls, the project files associated with the session you created are shown. If you run whoami, the command returns the cdsw user.

   Once you are connected, you should see something like this:

   ```
   $ cdswctl ssh-endpoint -p ml-at-scale -m 4 -c 2
   Forwarding local port 7847 to port 2222 on session bhsb7k4eqmonap62 in p
   roject finance/customerchurn.
   You can SSH to the session using

       ssh -p 7847 cdsw@localhost
   ```

5. Add an entry into your SSH config file.

   For example:

   ```
   $ cat ~/.ssh/config
   Host cdsw-public
       HostName localhost
       IdentityFile ~/.ssh/id_rsa
       User cdsw
       Port 7847
   ```

   HostName is always localhost and User is always cdsw. You get the Port number from Step 2.

Setting up VS Code

In VS Code, you can configure an SSH interpreter. Cloudera Machine Learning uses this method to connect to VS Code and act as its interpreter.

Before you begin

Ensure that you have installed the following:

- Remote SSH extension

  Remove Development using SSH
- [Optional] Python extension
- [Optional] R extension

About this task

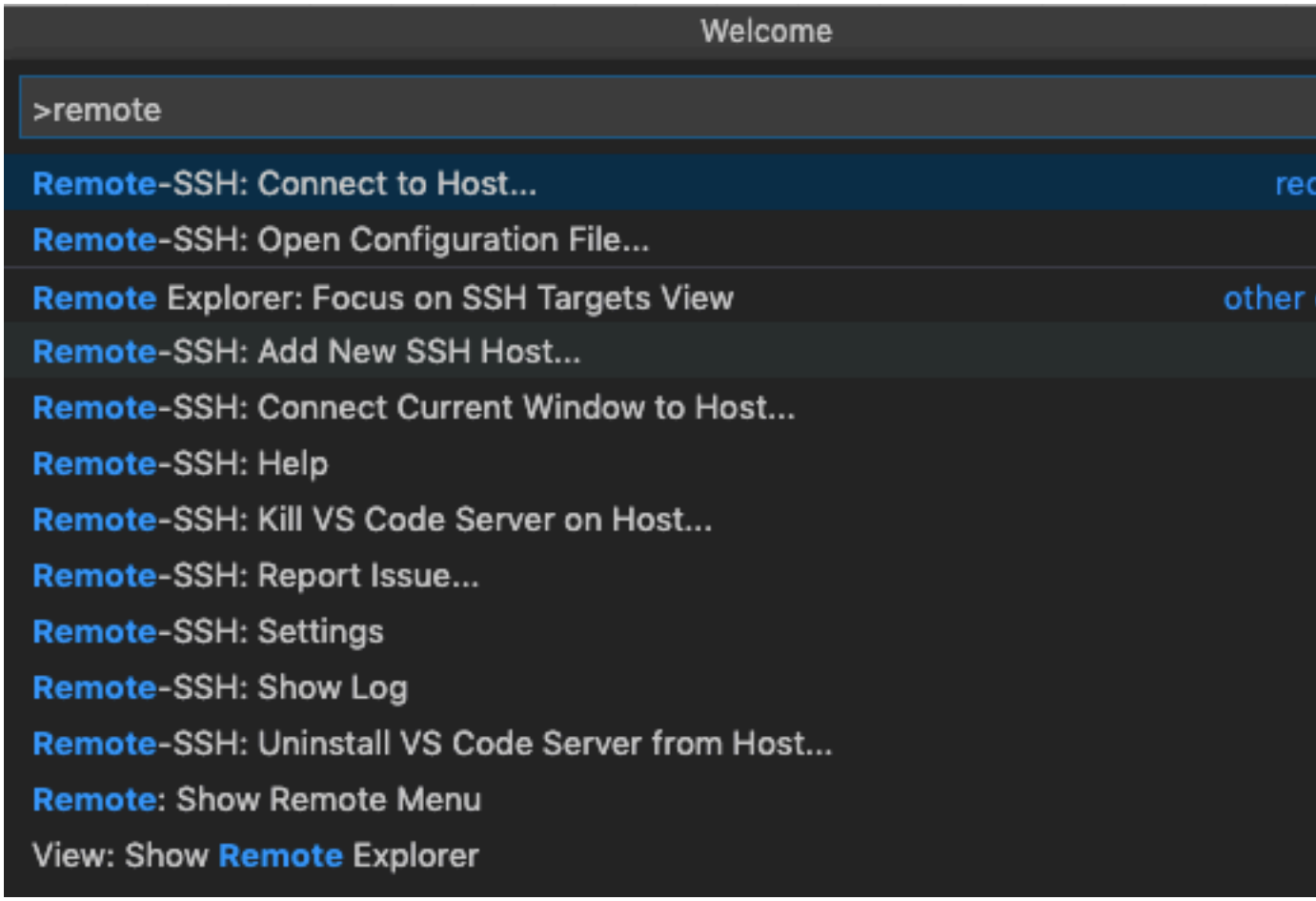
Before you begin, ensure that the SSH endpoint for Cloudera Machine Learning is running on your local machine. If additional information is required, see the documentation for your version of VS Code for specific instructions.

Procedure

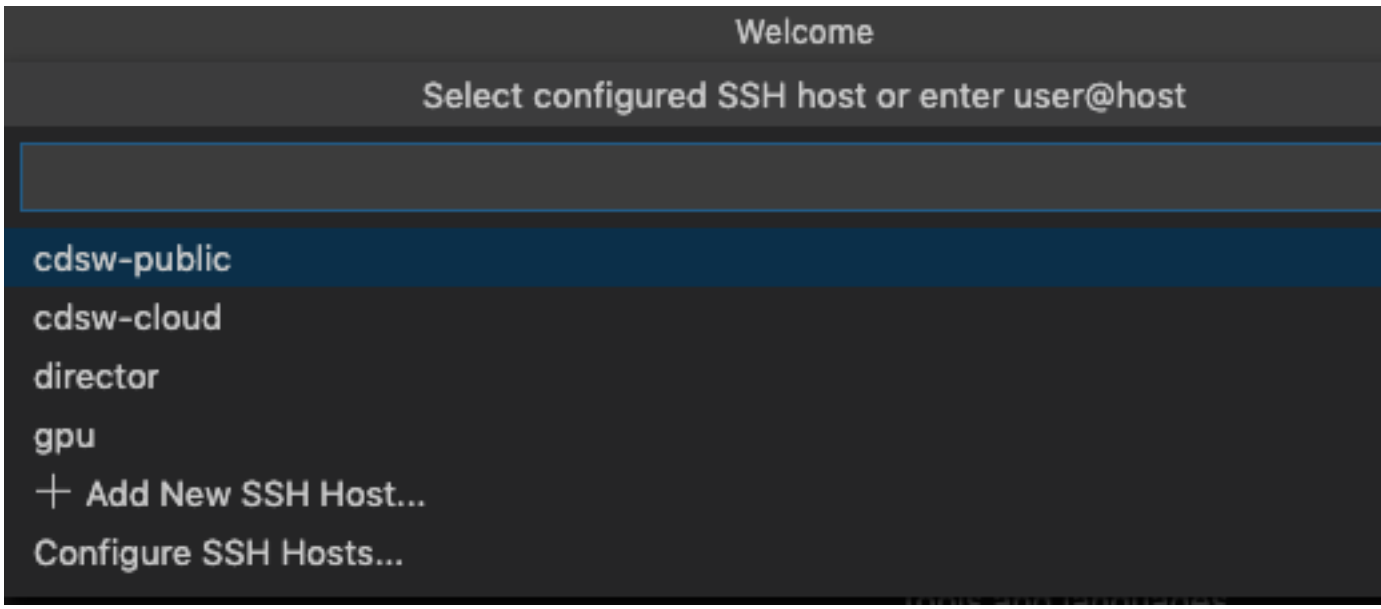
1. Verify that the SSH endpoint for Cloudera Machine Learning is running with cdswctl.

   If the endpoint is not running, start it.
2. Open VS Code.

3. Open the command pallet and connect to a remote host.

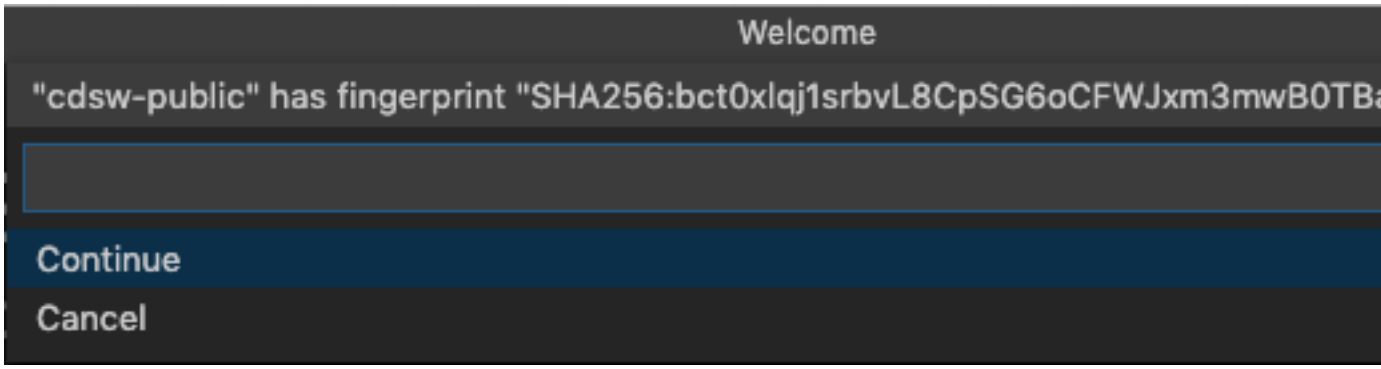


4. Connect to the host you added previously.



5. For the first connection, you must accept the fingerprint.

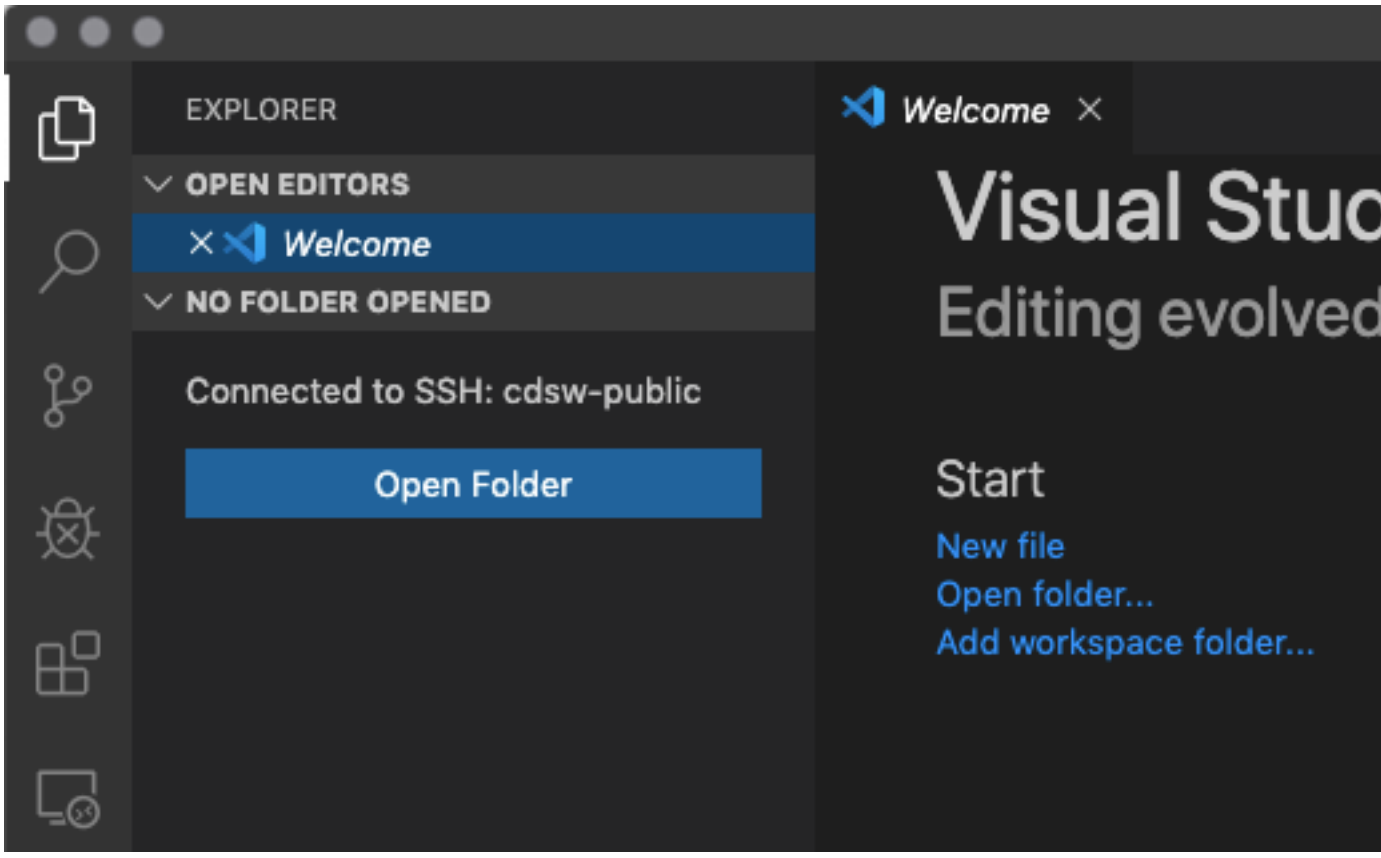
   You might not see a pop up, so pay attention to VS Code. If it's the first time your are connecting to a new session, or the port number changed, you will need to accept the fingerprint.



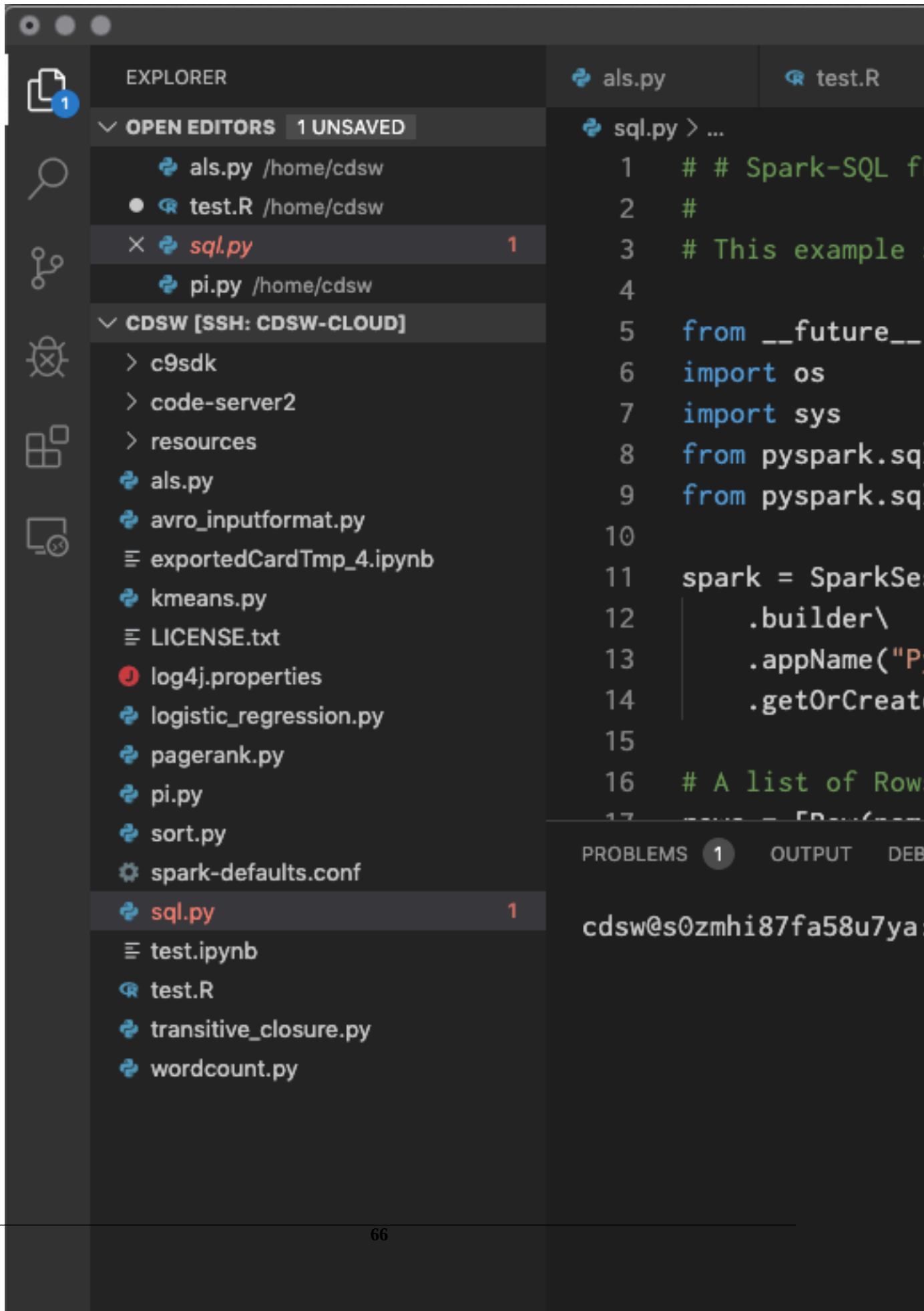
   While VS Code connects and sets up the remote connection, it installs some helper applications on the Cloudera Machine Learning server. Sometimes the remote session dies. Click Retry or if it's taking a long time, restart the remote session and it will recover.

   Note: If you get stuck in a loop during setup with VS Code reconnecting every 30 secs or so, the issue is with the lock file that VS Code creates during the install. Close VS Code and in CML terminal, delete the /home/cdsw/.vscode-server/ directory and start again.

6. After you are connected, you can open the Explorer and view and edit the files in the /home/cdsw directory.



7. From the Explorer view, you can edit any of the files on your Cloudera Machine Learning server.


EXPLORER
OPEN EDITORS 1 UNSAVED
als.py /home/cdsw
test.R /home/cdsw
sql.py 1
pi.py /home/cdsw
CDSW [SSH: CDSW-CLOUD]
c9sdk
code-server2
resources
als.py
avro_inputformat.py
exportedCardTmp_4.ipynb
kmeans.py
LICENSE.txt
log4j.properties
logistic_regression.py
pagerank.py
pi.py
sort.py
spark-defaults.conf
sql.py 1
test.ipynb
test.R
transitive_closure.py
wordcount.py
als.py
test.R
sql.py > ...
1 # # Spark-SQL f
2 #
3 # This example
4
5 from __future__
6 import os
7 import sys
8 from pyspark.sq
9 from pyspark.sq
10
11 spark = SparkSe
12 .builder\
13 .appName("P
14 .getOrCreat
15
16 # A list of Row
PROBLEMS 1
OUTPUT
DEB
cdsw@s0zmhi87fa58u7ya

Using the Explorer view, you remotely edit and modify your Cloudera Machine Learning files. VS Code also supports Python and R which you offer has some powerful coding tools that you can take advantage of over the remote connection.

(Optional) Using VS Code with Python

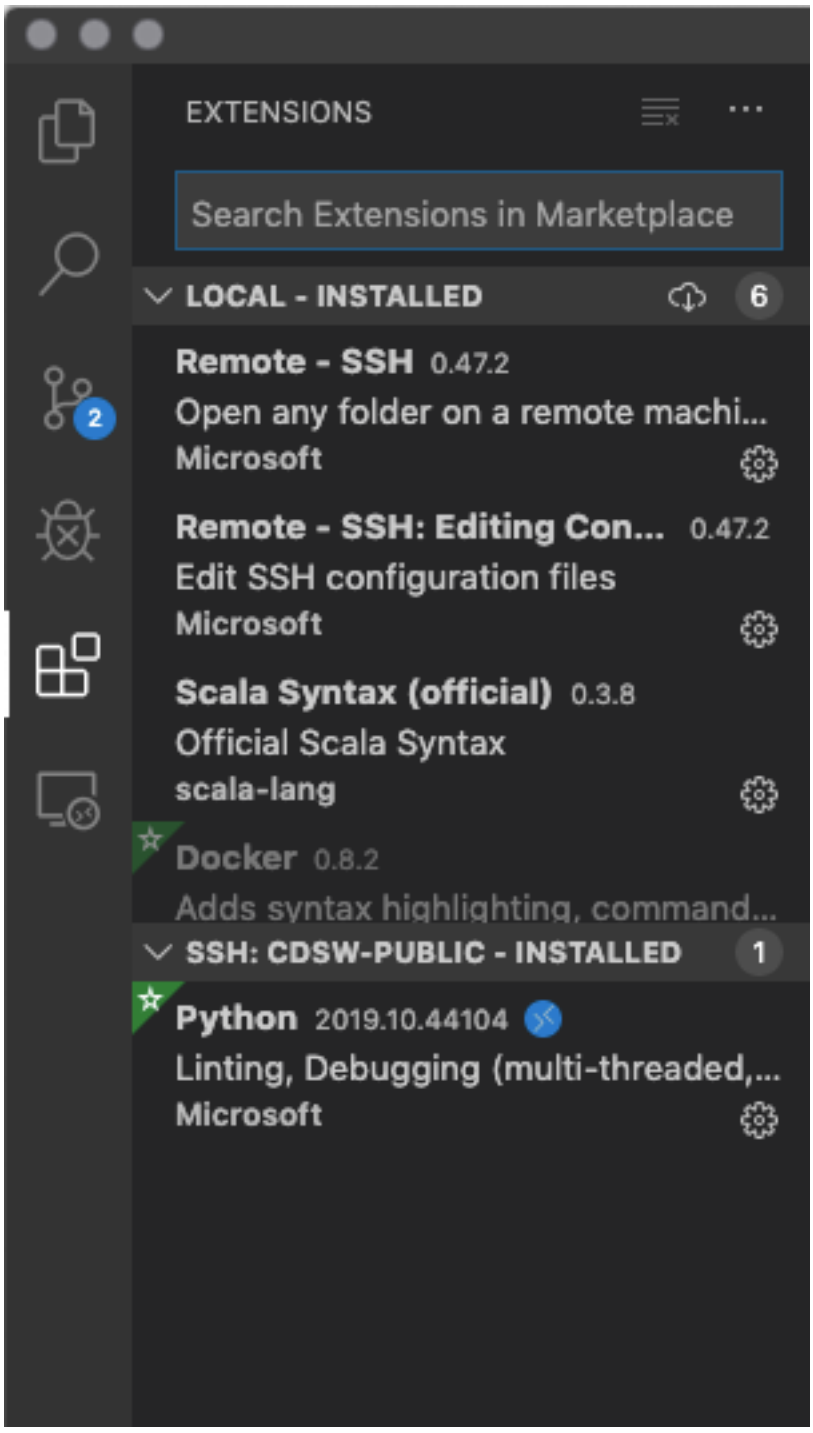
You can use VS Code with Python.

About this task

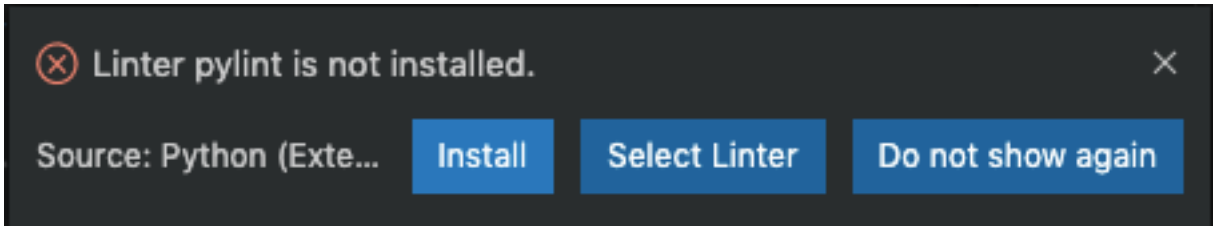
To take full advantage of VS Code Python tools, you must install the Python extension into the remote ssh session. You must install the extension the first time you connect a newly configured remote session.

Procedure

1. Install the Python extension.



2. With the Extension installed, once you open your first python file, you will be prompted to install pylint Linter.

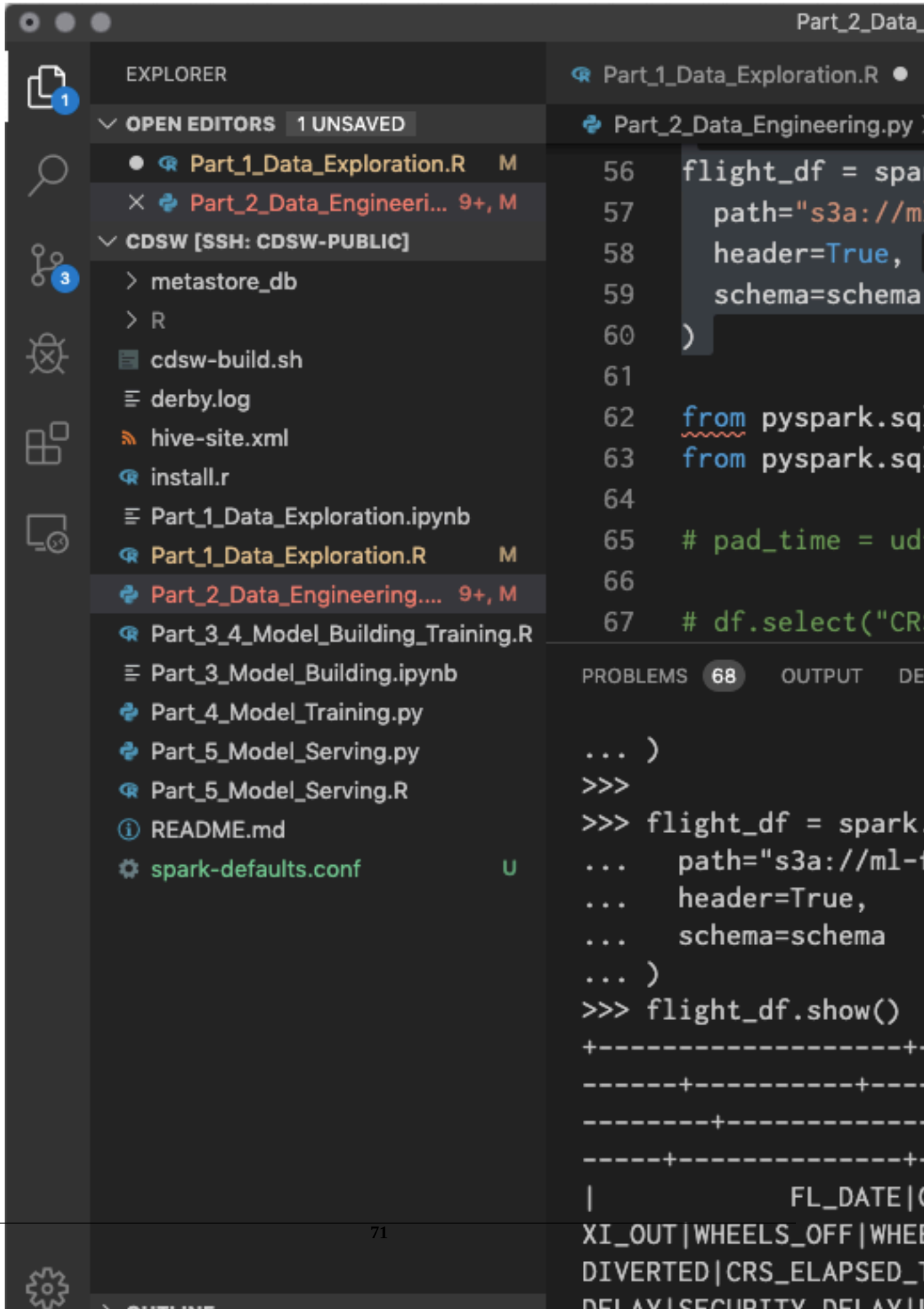


3. Click Install.

   VS Code opens a terminal and runs the code needed to install the linter. It's important to note that this is a remote terminal, running on an engine in Cloudera Machine Learning. It's the same as if you launched a terminal inside a running workbench.

4. If you want to run arbitrary Python code inside VS Code, open a Python file, select some code, right click, and select Run Selection/Line in Python Terminal.

   You can also just hit Shift-Enter in the code editor window. This will open up a new terminal if there isn't one and run the selected code. Since this is a remote session, you can run pyspark directly inside VS Code.


Part_2_Data_
EXPLORER
OPEN EDITORS 1 UNSAVED
Part_1_Data_Exploration.R M
Part_2_Data_Engineeri... 9+, M
CDSW [SSH: CDSW-PUBLIC]
metastore_db
R
cdsw-build.sh
derby.log
hive-site.xml
install.r
Part_1_Data_Exploration.ipynb
Part_1_Data_Exploration.R M
Part_2_Data_Engineering.... 9+, M
Part_3_4_Model_Building_Training.R
Part_3_Model_Building.ipynb
Part_4_Model_Training.py
Part_5_Model_Serving.py
Part_5_Model_Serving.R
README.md
spark-defaults.conf U
Part_1_Data_Exploration.R
Part_2_Data_Engineering.py
flight_df = spa
path="s3a://m
header=True,
schema=schema
)
from pyspark.sq
from pyspark.sq
pad_time = ud
df.select("CR
PROBLEMS 68 OUTPUT DE
...)
>>>
>>> flight_df = spark
... path="s3a://ml-
... header=True,
... schema=schema
...)
>>> flight_df.show()
+----------+
------+----------+---
--------+----------
-----+----------+
| FL_DATE|
XI_OUT|WHEELS_OFF|WHE
DIVERTED|CRS_ELAPSED_
DELAY|SECURITY_DELAY|

For more complex code requirements, you can also use the Python Debugging feature in VS Code.

(Optional) Using VS Code with R

You can use VS Code with R.

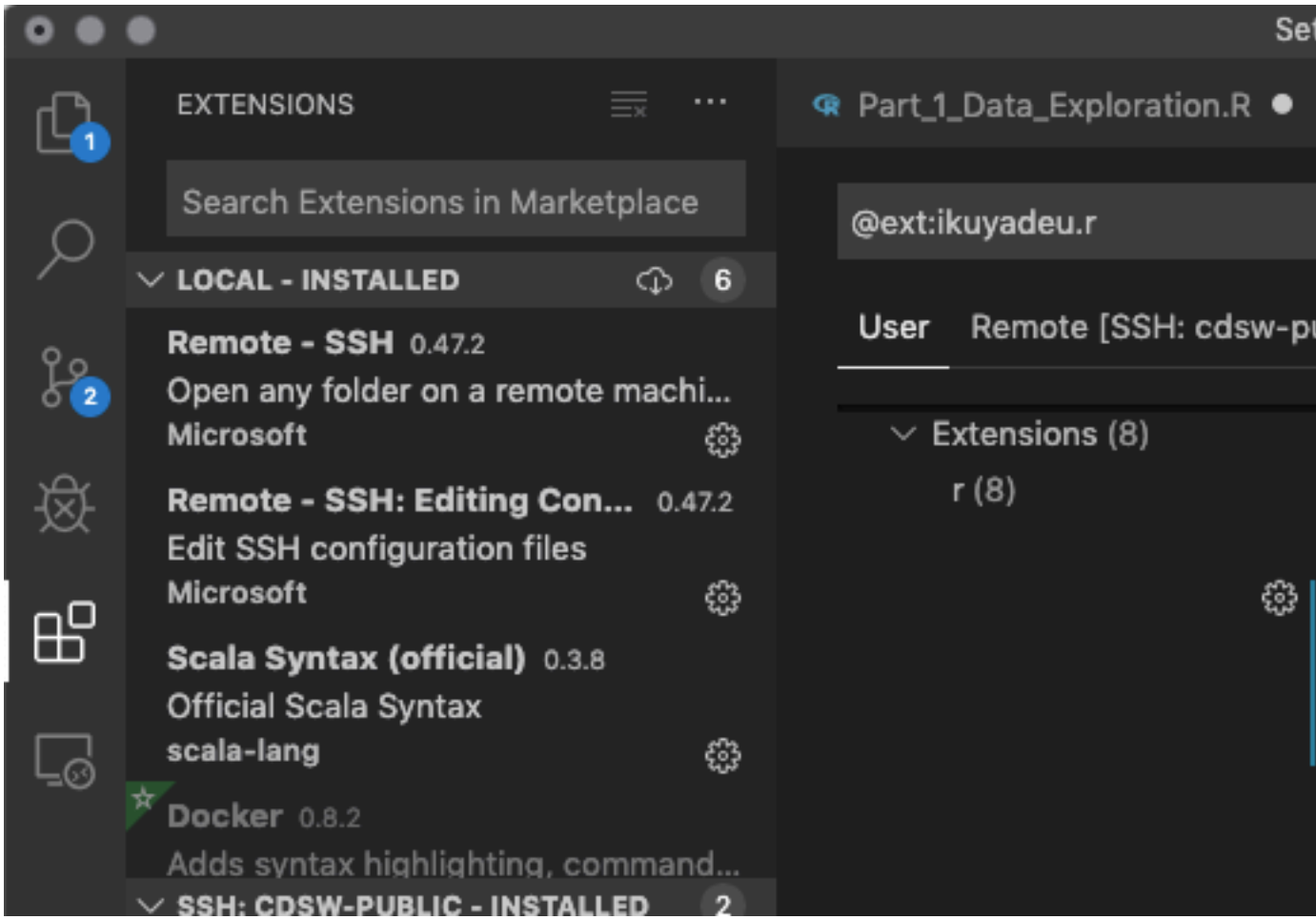
About this task

The R extension provides similar capabilities as Python. This means you can edit R files with code completion and execute arbitrary code in the terminal. With sparklyr, you can run spark code using R inside VS Code.

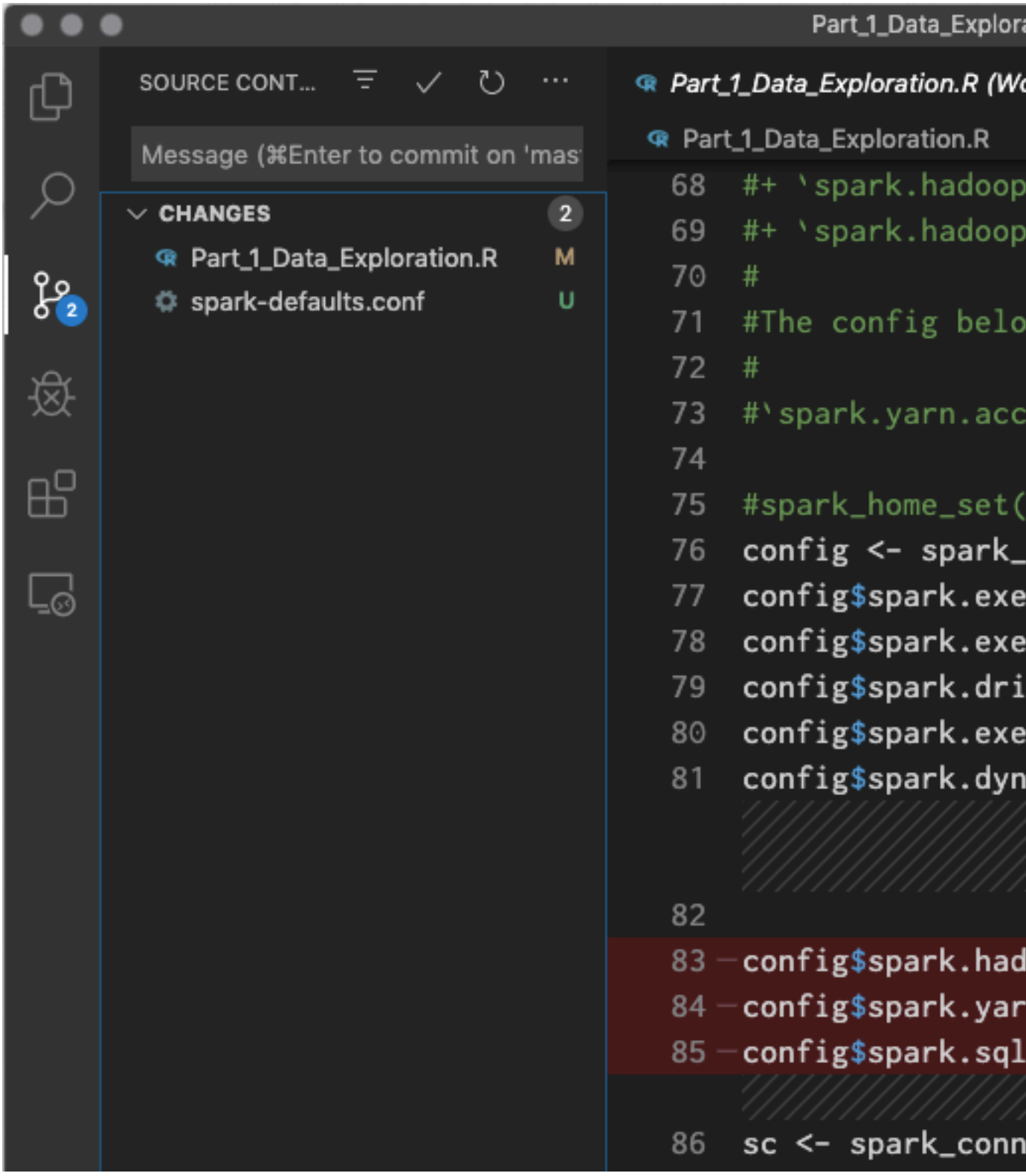
Procedure

1. Prior to installing the R extension, check where your R binary lives in CDW by running which R and then pasting that into the R > Rterm: Linux setting in VS Code.

   The R binary is most likely located in /usr/local/bin/R, but its best to check.



2. After you install the R extension, you can use sparklyr to run spark code using R inside VS Code.



(Optional) Using VS Code with Jupyter

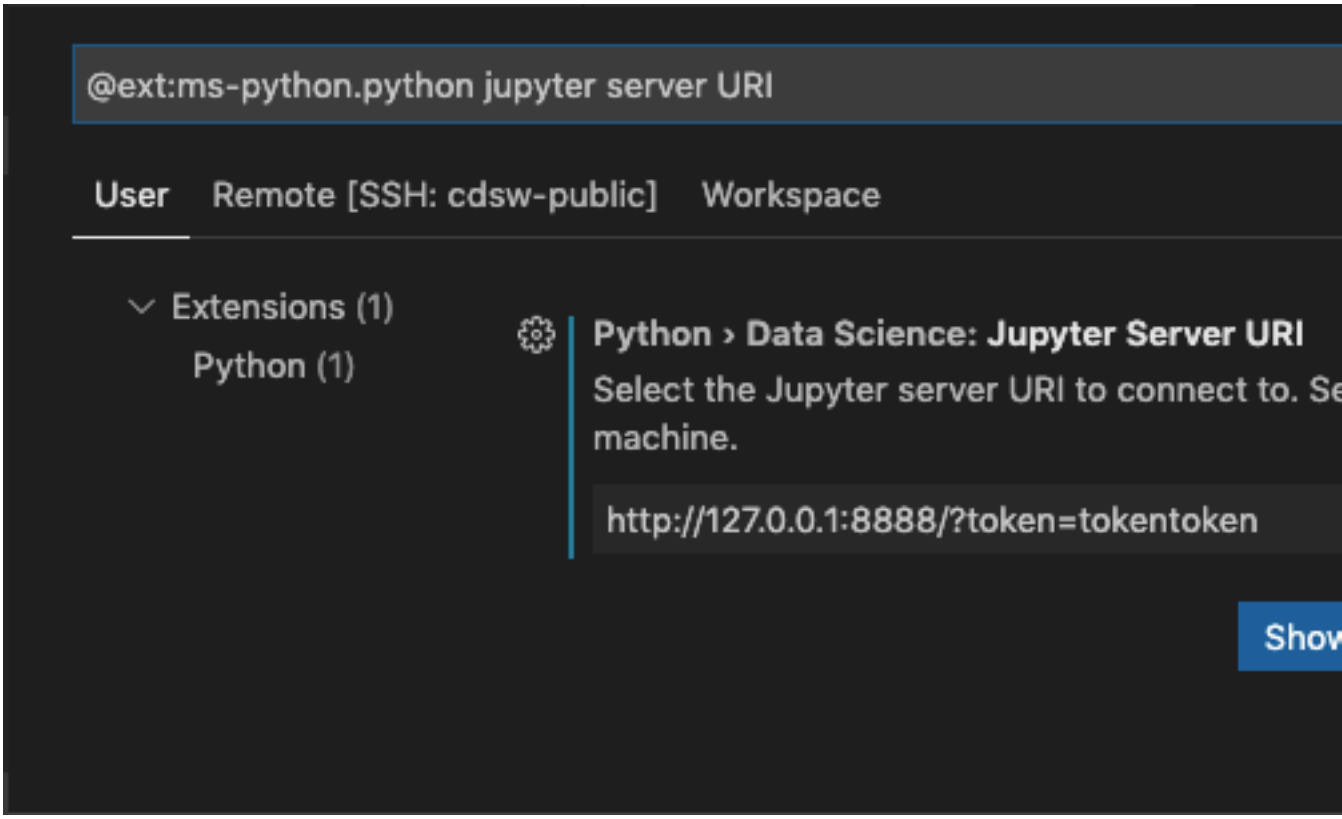
You can use VS Code with Jupyter Notebooks.

About this task

You can work on Jupyter Notebooks within VS Code. This gives you all the great code completion, syntax highlighting and documentation hints that are part of the VS Code experience and the interactivity of a Jupyter Notebook. Any changes you make to the Notebook will be reflected on the CDSW / CML server and can be viewed online using Jupyter Notebook as a browser based editor.

Procedure

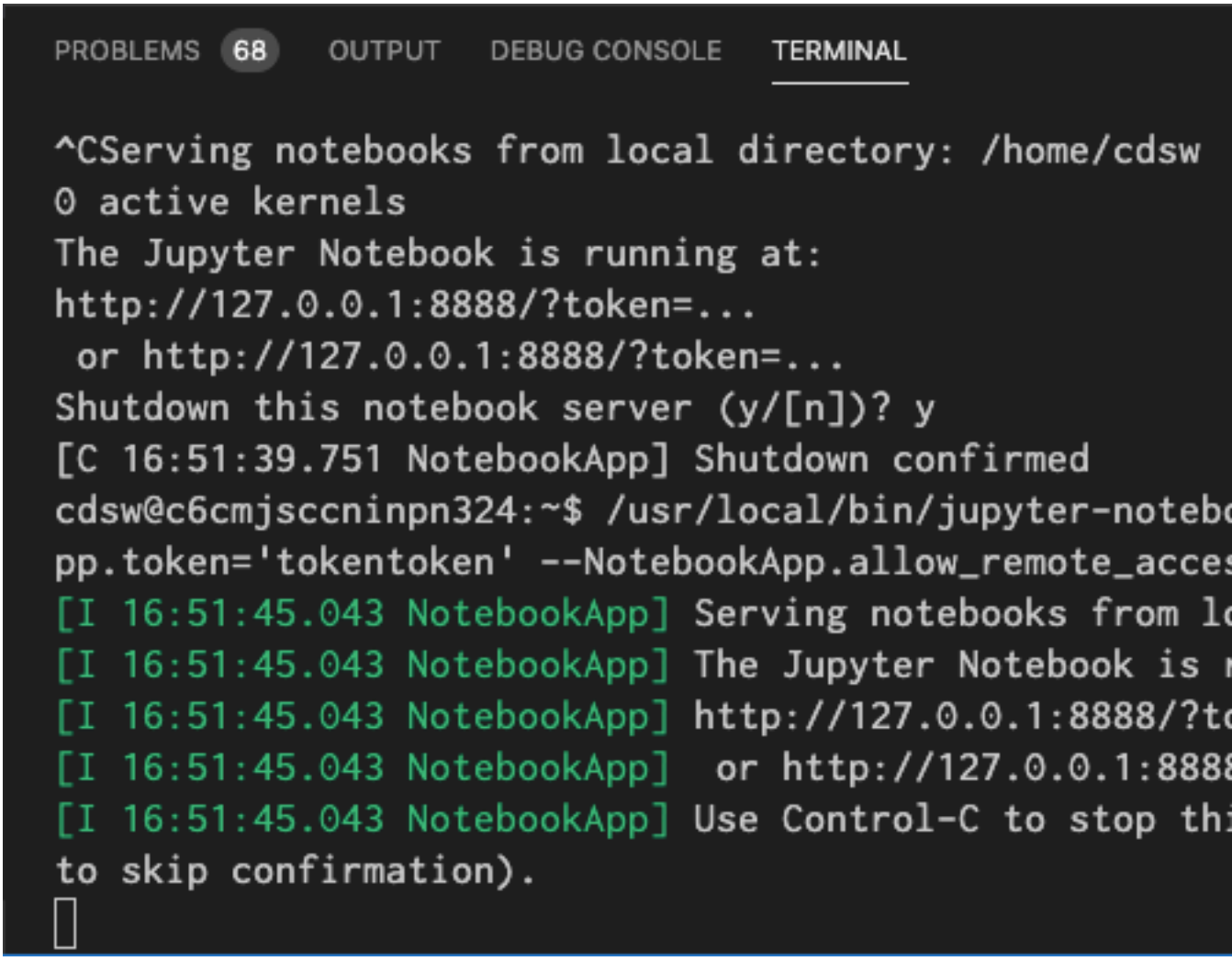
1. Because of the way CML uses the internal networking and port forwarding of Kubernetes, when VS Code launches a Jupyter Server it binds to the wrong address and access is blocked. You therefore have to launch your own Jupyter Server and tell VS code to connect to that.
   a) The first setting you need to set is the Python > Data Science: Jupyter Server URI setting. Set this to http://127.0.0.1:8888/?token=[some-token].



   b) Then you need to open a terminal to launch a Jupiter Notebook server.

   You can launch it using: /usr/local/bin/jupyter-notebook —no-browser —ip=127.0.0.1 —NotebookApp.token=[some-token] —NotebookApp.allow_remote_access=True .

   This creates a Jupyter server that any new Notebooks you launch will run in.


PROBLEMS 68
OUTPUT
DEBUG CONSOLE
TERMINAL
^CServing notebooks from local directory: /home/cdsw
0 active kernels
The Jupyter Notebook is running at:
http://127.0.0.1:8888/?token=...
or http://127.0.0.1:8888/?token=...
Shutdown this notebook server (y/[n])? y
[C 16:51:39.751 NotebookApp] Shutdown confirmed
cdsw@c6cmjsccninpn324:~$ /usr/local/bin/jupyter-noteb
pp.token='tokentoken' --NotebookApp.allow_remote_acce
[I 16:51:45.043 NotebookApp] Serving notebooks from l
[I 16:51:45.043 NotebookApp] The Jupyter Notebook is
[I 16:51:45.043 NotebookApp] http://127.0.0.1:8888/?t
[I 16:51:45.043 NotebookApp] or http://127.0.0.1:888
[I 16:51:45.043 NotebookApp] Use Control-C to stop th
to skip confirmation).

2. After you install the Jupyter Notebooks, you can use it inside VS Code.

Part_1_Data_Exploration.ipynb* — cdsw [SSH: cdsw-public]

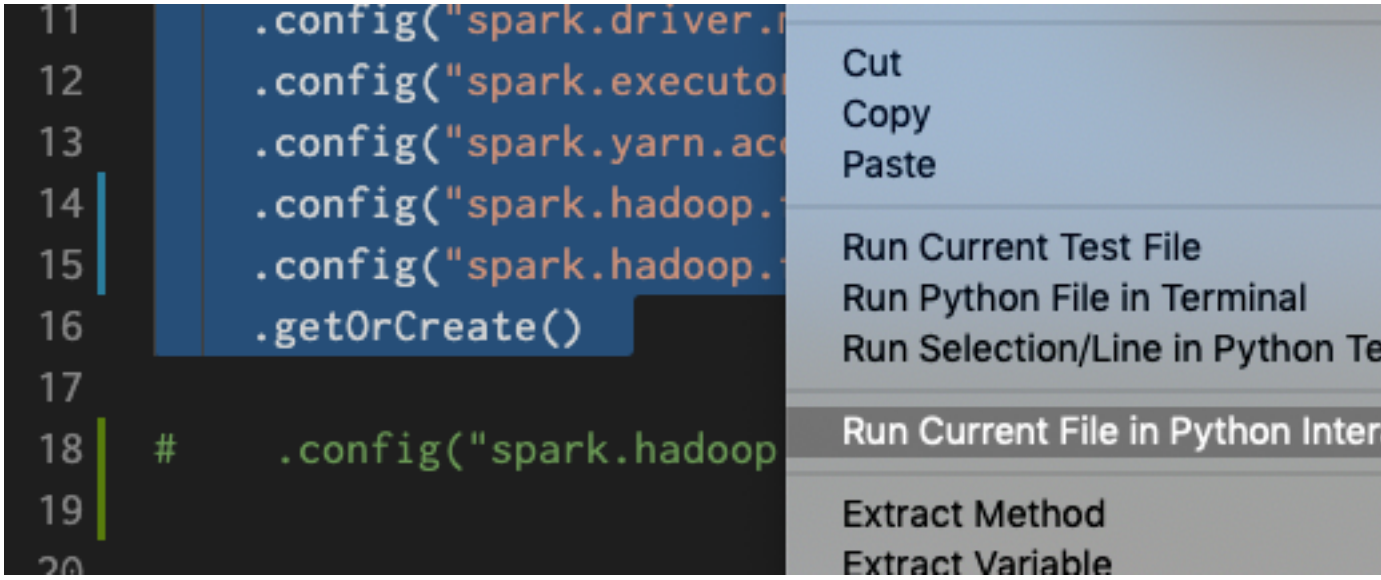
Part_1_Data_Exploration.ipynb*

Plot this using a Tufte-like layout :)

```
[7] sns.set_style("white",{'axes.axisbelow': False
    plt.bar(
    cancel_by_year_percent_pd.year,
    cancel_by_year_percent_pd.delay_percent,
    align='center',
    alpha=0.5,
    color='#888888',
    )
    plt.grid(color='#FFFFFF', linestyle='-', linew
    plt.title(
    'Percentage Cancelled Flights by Year',
    color='grey'
    )
    plt.xticks(
    cancel_by_year_percent_pd.year,
    color='grey'
    )
    plt.yticks(color='grey')
    sns.despine(left=True,bottom=True)
```

3. Another feature that you can use with VS Code is running a temporary Notebook for executing random code snippets. Select code you want to run, right click and click Run Current File in Python Interactive Window.

   This is less robust though and will create many Untitled*.ipynb files in your home directory.

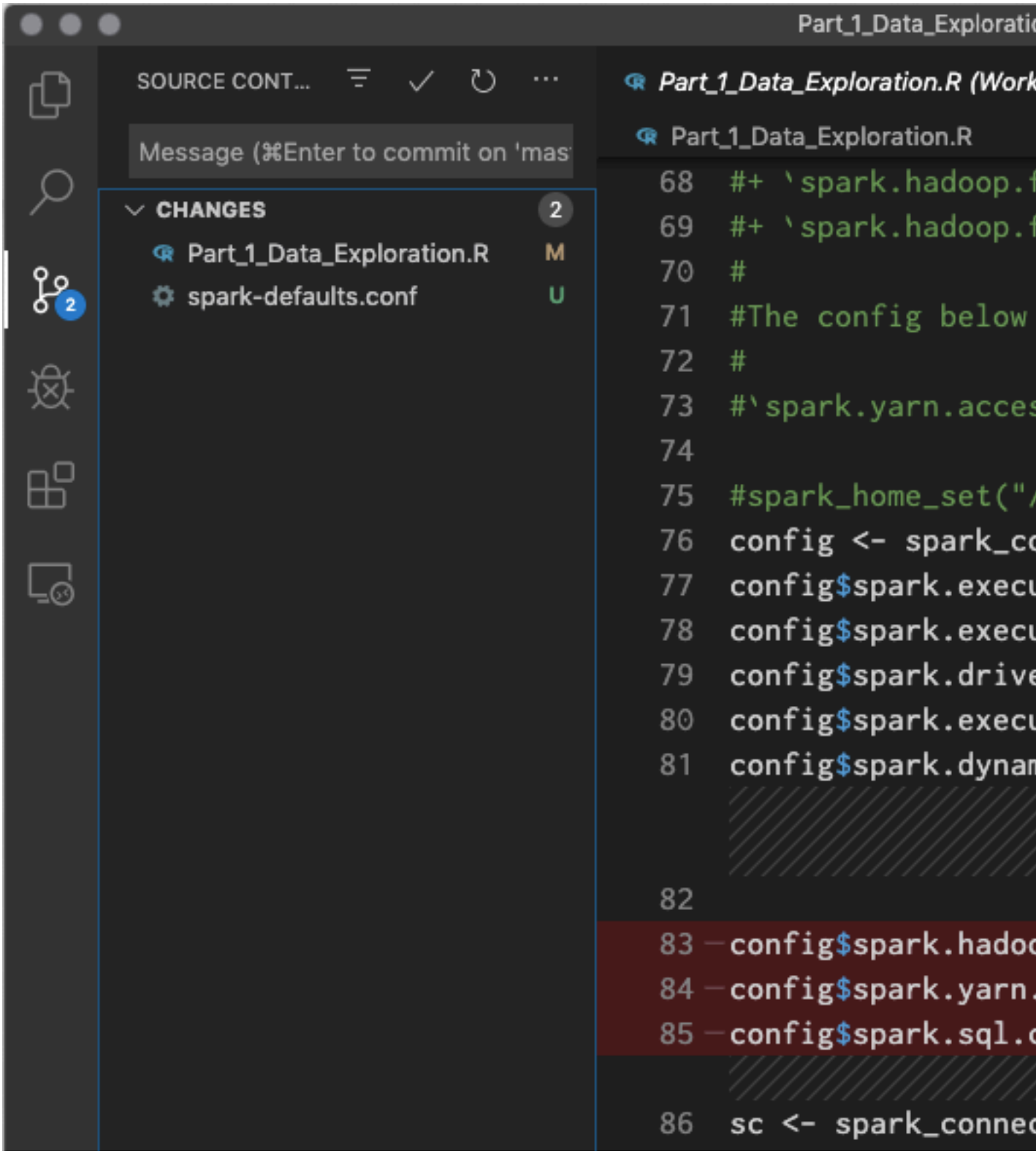


(Optional) Using VS Code with Git integration

VS Code has substantial Git integration.

About this task

If you created your project from a git repo or a custom template, your changes and outside changes made to the repo will automatically appear.



Limiting files in Explorer view

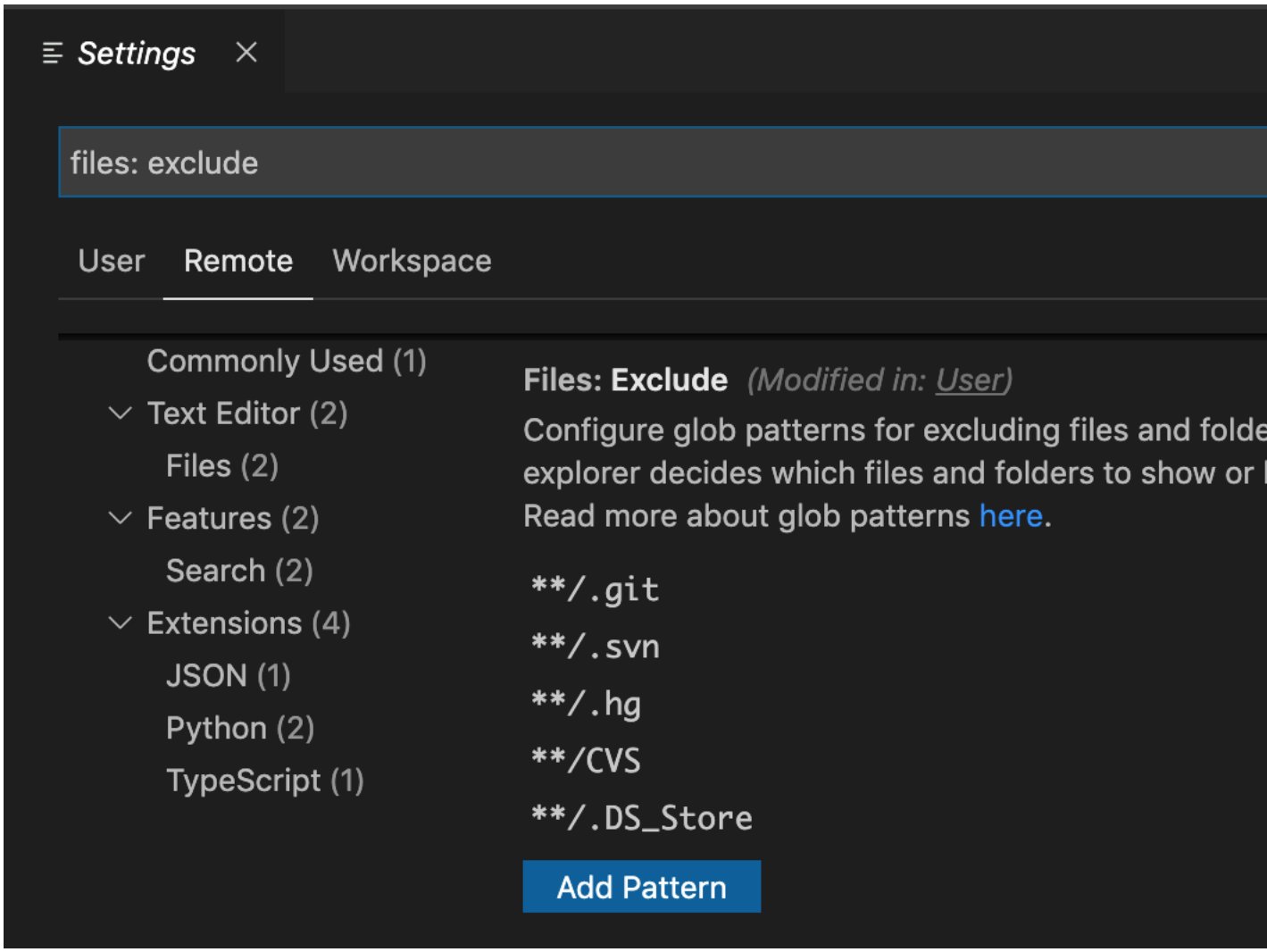
You can limit the number of files shown in the Explorer view.

About this task

If you end up with many of .[something] directories, in /home/cdsw, it can be difficult to navigate.

Procedure

If you add the **/.* pattern to the Files: Exclude setting, it will hide all those files and directories.



Git for Collaboration

Cloudera Machine Learning provides seamless access to Git projects. Whether you are working independently, or as part of a team, you can leverage all of benefits of version control and collaboration with Git from within Cloudera Machine Learning.

Teams that already use Git for collaboration can continue to do so. Each team member will need to create a separate Cloudera Machine Learning project from the central Git repository. For anything but simple projects, Cloudera recommends using Git for version control. You should work on Cloudera Machine Learning the same way you would work locally, and for most data scientists and developers that means using Git.

Cloudera Machine Learning does not include significant UI support for Git, but instead allows you to use the full power of the command line. If you launch a session and open a Terminal, you can run any Git command, including `init`, `add`, `commit`, `branch`, `merge` and `rebase`. Everything should work exactly as it does locally.

When you create a project, you can optionally supply an HTTPS or SSH Git URL that points to a remote repository. The new project is a clone of that remote repository. You can commit, push and pull your code by running a console and opening a Terminal. Note that if you want to use SSH to clone the repo, you will need to first add your personal Cloudera Machine Learning SSH key to your GitHub account. For instructions, see *Adding SSH Key to GitHub*.

If you see Git commands hanging indefinitely, check with your cluster administrators to make sure that the SSH ports on the Cloudera Machine Learning hosts are not blocked.

Related Information

Adding an SSH Key to GitHub
Creating a Project

Linking an Existing Project to a Git Remote

If you did not create your project from a Git repository, you can link an existing project to a Git remote (for example, git@github.com:username/repo.git) so that you can push and pull your code.

Procedure

1. Launch a new session.
2. Open a terminal.
3. Enter the following commands:

   Shell

   ```
   git init
   git add *
   git commit -a -m 'Initial commit'
   git remote add origin git@github.com:username/repo.git
   ```

   You can run git status after git init to make sure your .gitignore includes a folder for libraries and other non-code artifacts.

Web Applications Embedded in Sessions

This topic describes how Cloudera Machine Learning allows you to embed web applications for frameworks such as Spark 2, TensorFlow, Shiny, and so on within sessions and jobs.

Many data science libraries and processing frameworks include user interfaces to help track progress of your jobs and break down workflows. These are instrumental in debugging and using the platforms themselves. For example, TensorFlow visualizations can be run on TensorBoard. Other web application frameworks such as Shiny and Flask are popular ways for data scientists to display additional interactive analysis in the languages they already know.

Cloudera Machine Learning allows you to access these web UIs directly from sessions and jobs. This feature is particularly helpful when you want to monitor and track progress for batch jobs. Even though jobs don't give you access to the interactive workbench console, you can still track long running jobs through the UI. However, note that the UI is only active so long as the job or session is active. If your session times out after 60 minutes (default timeout value), so will the UI.

Important: If you want to share your web application as a long-running standalone application that other business users can access, Cloudera recommends you now use the Applications feature to support long-running web applications on ML workspaces.

If you are only running a server-backed visualization as part of your own analysis, then you can continue to keep embedding web applications in sessions as described in this topic. Note that running web applications in sessions is also the recommended way to develop, test, and debug analytical apps before deployment.

CDSW_APP_PORT and CDSW_READONLY_PORT are environment variables that point to general purpose public ports. Any HTTP services running in containers that bind to CDSW_APP_PORT or CDSW_READONLY_PORT are available in browsers at: http://<$CDSW_ENGINE_ID>.<$CDSW_DOMAIN>. Therefore, TensorBoard, Shiny, Flask or any other web framework accompanying a project can be accessed directly from within a session or job, as long as it is run on CDSW_APP_PORT or CDSW_READONLY_PORT.

CDSW_APP_PORT is meant for applications that grant some level of control to the project, such as access to the active session or terminal. CDSW_READONLY_PORT must be used for applications that grant read-only access to project results.

To access the UI while you are in an active session, click the grid icon in the upper right hand corner of the Cloudera Machine Learning web application, and select the UI from the dropdown. For a job, navigate to the job overview page and click the History tab. Click on a job run to open the session output for the job. You can now click the grid icon in the upper right hand corner of the Cloudera Machine Learning web application to access the UI for this session.

Limitations with port availability

Cloudera Machine Learning exposes only one port per-access level. This means, in version 1.6.0, you can run a maximum of 3 web applications simultaneously:

- one on CDSW_APP_PORT, which can be used for applications that grant some level of control over the project to Contributors and Admins,
- one on CDSW_READONLY_PORT, which can be used for applications that only need to give read-only access to project collaborators,
- and, one on the now-deprecated CDSW_PUBLIC_PORT, which is accessible by all users.

However, by default the editors feature runs third-party browser-based editors on CDSW_APP_PORT. Therefore, for projects that are already using browser-based third-party editors, you are left with only 2 other ports to run applications on: CDSW_READONLY_PORT and CDSW_PUBLIC_PORT. Keep in mind the level of access you want to grant users when you are selecting one of these ports for a web application.

Example: A Shiny Application

This example demonstrates how to create and run a Shiny application and view the associated UI while in an active session.

Create a new, blank project and run an R console. Create the files, ui.R and server.R, in the project, and copy the contents of the following example files provided by Shiny by RStudio:

R

```
# ui.R

library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

R

```
# server.R
library(shiny)
# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should re-execute automatically
  #     when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```

Run the following code in the interactive workbench prompt to install the Shiny package, load the library into the engine, and run the Shiny application.

R

```
install.packages('shiny')

library('shiny')

runApp(port=as.numeric(Sys.getenv("CDSW_READONLY_PORT")), host="127.0.0.1",
 launch.browser="FALSE")
```

Finally, click the grid icon in the upper right hand corner of the Cloudera Machine Learning web application, and select the Shiny UI, Hello Shiny!, from the dropdown. The UI will be active as long as the session is still running.

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.

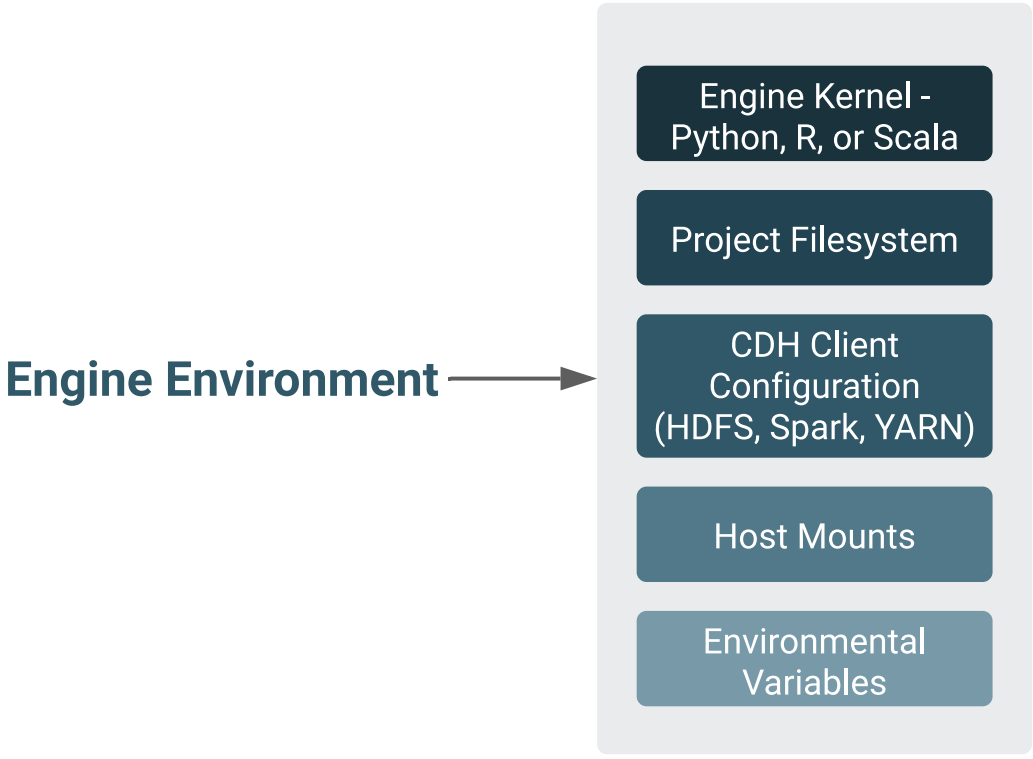
New versions of the base engine image are released periodically. However, existing projects are not automatically upgraded to use new engine images. Older images are retained to ensure you are able to test code compatibility with the new engine before upgrading to it manually.

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.

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, 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.

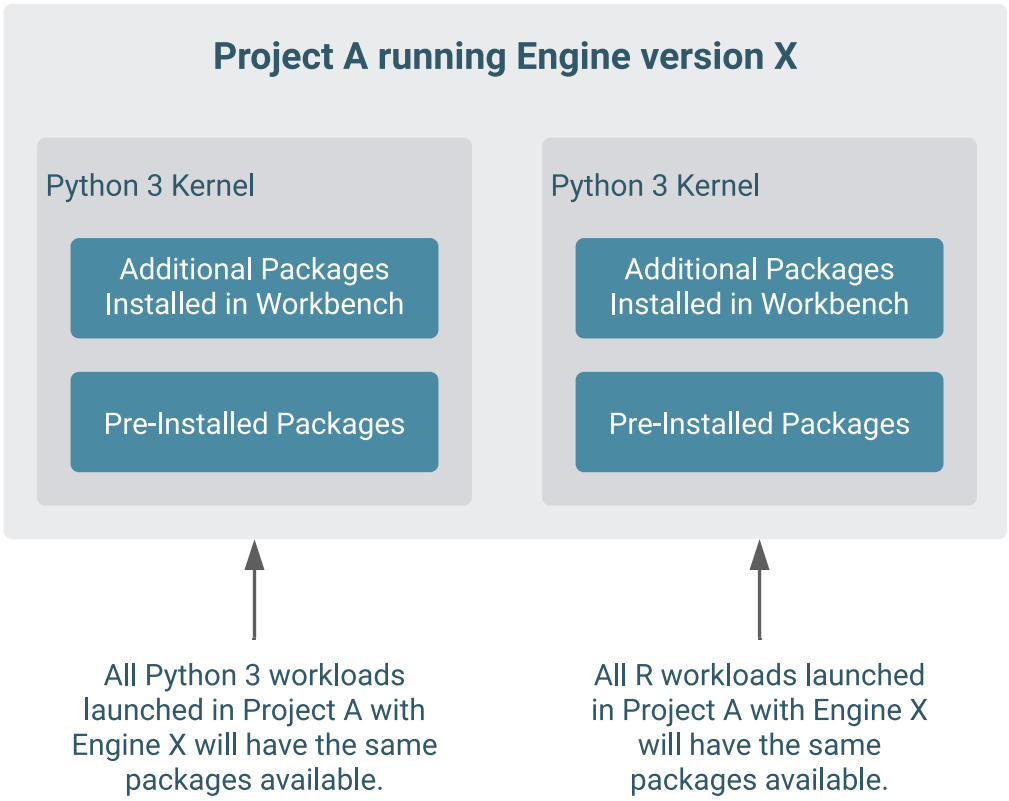
If you accidentally install these packages (or you see unexpected behavior when you switch a project from Legacy Engine to Runtimes), the simplest solution is to delete /home/cdsw/.local/lib/python and reinstall your project’s dependencies from the project overview page.

Engine Dependencies

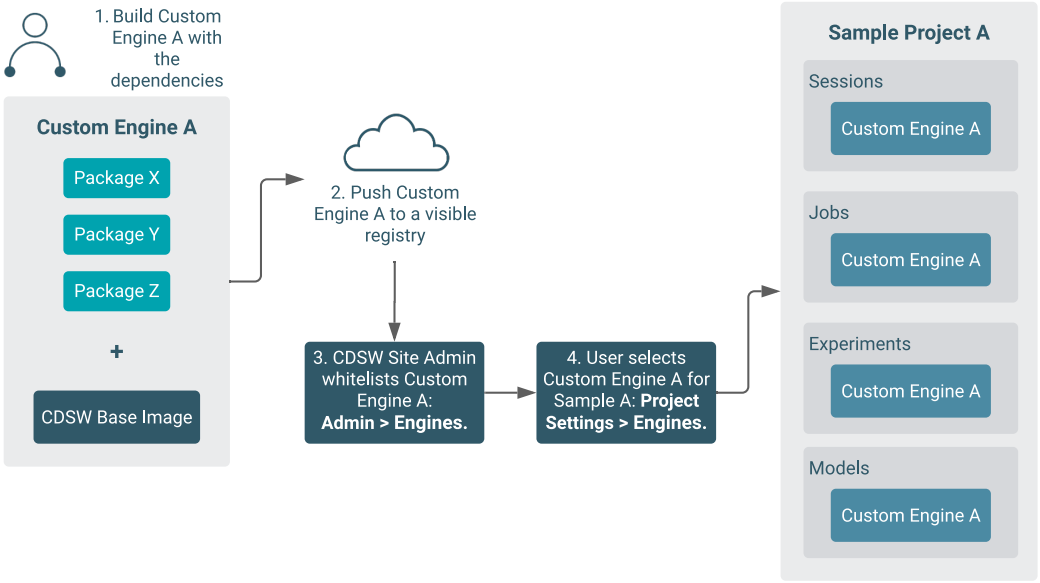
This topic describes the options available to you for mounting a project's dependencies into its engine environment. Depending on your projects or user preferences, one or more of these methods may be more appropriate for your deployment.

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*.

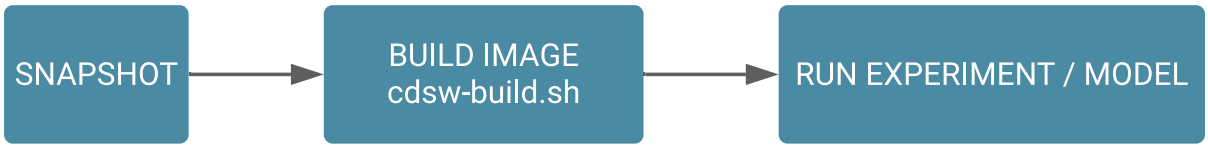
Related Information

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 5: 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 6: 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 7: 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 8: 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.

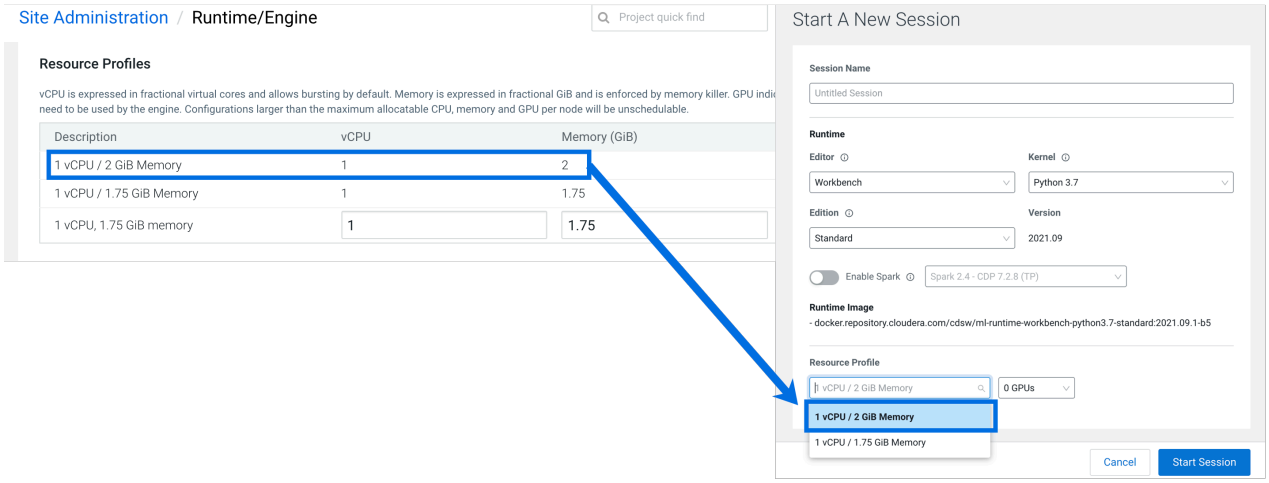
2. Add a new profile under Resource Profiles.

   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 Cloudera Machine Learning hosts are equipped with GPUs, and you have enabled them for use by setting the relevant properties in cdsw.conf.

Results

Figure 9: 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/cdsw/.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 COND A_DEFAULT_ENV and CONDA_PREFIX environment variables to point to Conda environments under /home/cd sw/.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-pack
   ages
   ```

   Python 3

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

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.

Environment Variable	Description
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 *Installing Additional Packages*.

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.
CDSW_PUBLIC_PORT	**Note:** This property is deprecated. See CDSW_APP_PORT and CDSW_READONLY_PORT for alternatives. A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_PUBLIC_PORT will be available in browsers at: http(s)://read-only-<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. By default, CDSW_PUBLIC_PORT is set to 8080. A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera Machine Learning web application, as long as the job or session is still running. For more details, see *Accessing Web User Interfaces from Cloudera Machine Learning*. In Cloudera Machine Learning, setting CDSW_PUBLIC_PORT to a non-default port number is not supported.

Environment Variable	Description
CDSW_APP_PORT	A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_APP_PORT will be available in browsers at: http(s)://read-only-<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. Use this port for applications that grant some control to the project, such as access to the session or terminal. A direct link to these web services will be available from the grid icon in the upper right corner of the Cloudera 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 *Accessing Web User Interfaces from Cloudera Machine Learning*. Note that if the Site Administrator has enabled Allow only session creators to run commands on active sessions, then the UI is only available to the session creator. Other users will not be able to access it. Use 127.0.0.1 as the IP.
CDSW_READONLY_PORT	A port on which you can expose HTTP services in the engine to browsers. HTTP services that bind CDSW_READONLY_PORT will be available in browsers at: http(s)://read-only-<*$CDSW_ENGINE_ID*>.<*$CDSW_DOMAIN*>. Use this port for applications that grant read-only access to project results. A direct link to these web services will be available to users with from the grid icon in the upper right corner of the Cloudera 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 *Accessing Web User Interfaces from Cloudera Machine Learning*. Use 127.0.0.1 as the IP.
CDSW_DOMAIN	The domain on which Cloudera Machine Learning is being served. This can be useful for iframing services, as demonstrated in *Accessing Web User Interfaces from Cloudera Machine Learning*.
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 paywall.

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 Maching 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-na
me>:<tag>
```

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

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

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/cdsw/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 /v
ar/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.

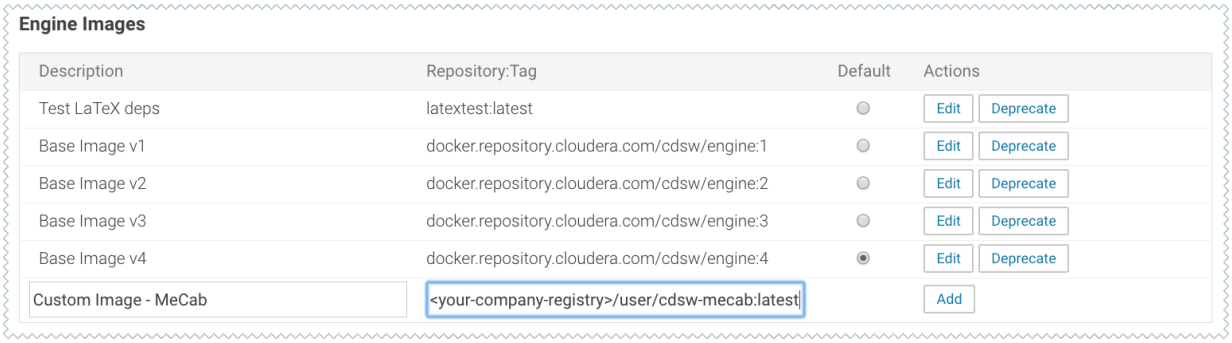
```
docker build --network=host -t <company-registry>/user/cdsw-mecab:latest .
 -f Dockerfile
```

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

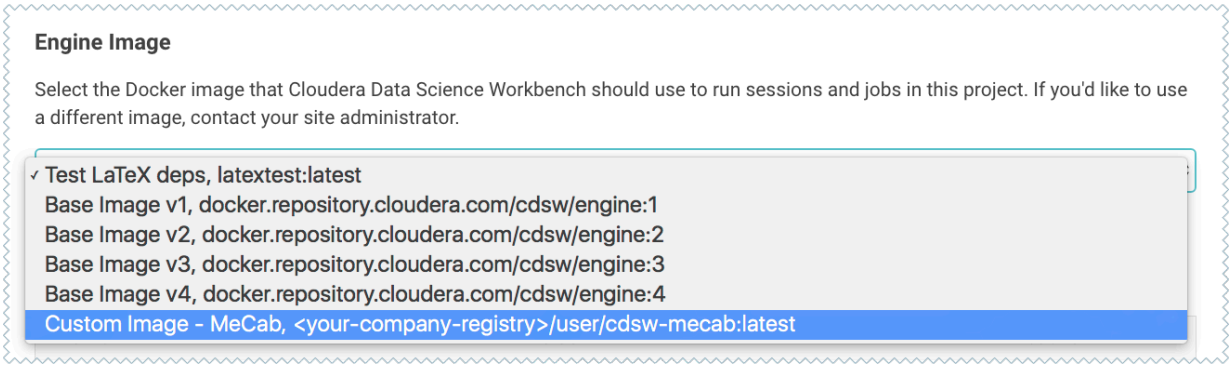
```
docker push <your-company-registry>/user/cdsw-mecab:latest
```

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

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



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

Library	Version
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
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

Package	Version
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

Library	Version
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
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
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

Package	Version
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 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
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

Package	Version
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 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

Library	Version
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
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

Package	Version
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

Package	Version
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
devtools	2.0.2

Related Information

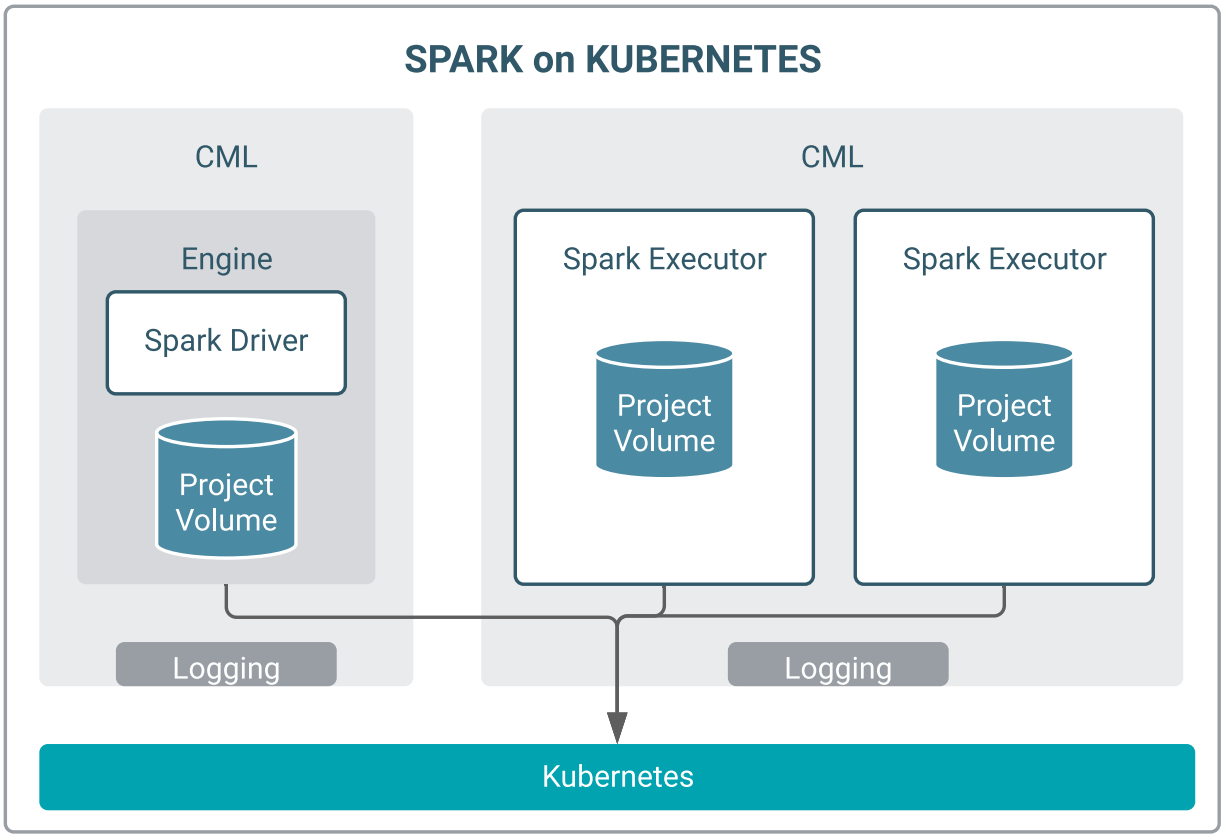
Base Engine 10
Base Engine 11
Base Engine 12
Base Engine 13
Base Engine 14

Apache Spark 2 and Spark 3 on CML

Apache Spark is a general purpose framework for distributed computing that offers high performance for both batch and stream processing. It exposes APIs for Java, Python, R, and Scala, as well as an interactive shell for you to run jobs.

In Cloudera Machine Learning (CML), Spark and its dependencies are bundled directly into the CML engine Docker image.

CML supports fully-containerized execution of Spark workloads via Spark's support for the Kubernetes cluster backend. Users can interact with Spark both interactively and in batch mode.



Dependency Management: In both batch and interactive modes, dependency management, including for Spark executors, is transparently managed by CML and Kubernetes. No extra required configuration is required. In interactive mode, CML leverages your cloud provider for scalable project storage, and in batch mode, CML manages dependencies though container images.

Autoscaling: CML also supports native cloud autoscaling via Kubernetes. When clusters do not have the required capacity to run workloads, they can automatically scale up additional nodes. Administrators can configure auto-scaling upper limits, which determine how large a compute cluster can grow. Since compute costs increase as cluster size increases, having a way to configure upper limits gives administrators a method to stay within a budget. Autoscaling policies can also account for heterogeneous node types such as GPU nodes.

Dynamic Resource Allocation: If a Spark job requires increasing memory or CPU resources as it executes a job, Spark can automatically increase the allocation of these resources. Likewise, the resources are automatically returned to the cluster when they are no longer needed. This mechanism is especially useful when multiple applications are sharing the resources of a cluster.

Workload Isolation: In CML, each project is owned by a user or team. Users can launch multiple sessions in a project. Workloads are launched within a separate Kubernetes namespace for each user, thus ensuring isolation between users at the K8s level.

Observability: Monitoring of Spark workloads, such as resources being consumed by Spark executors, can be performed using Grafana dashboards. For more information, see *Monitoring and Alerts* and *Monitoring ML Workspaces*.

Related Information

Monitoring and Alerts

Monitoring ML Workspaces

Apache Spark supported versions

Spark 2.4.7 is supported by Engines. Spark 2.4.8 and Spark 3.2.0 are available through Runtime Addons that can be selected when starting a session.

Note: Spark 3 does not work with Scala runtimes.

Spark Configuration Files

Cloudera Machine Learning supports configuring Spark 2 and Spark 3 properties on a per project basis with the spar k-defaults.conf file. If there is a file called spark-defaults.conf in your project root, this will be automatically be added to the global Spark defaults.

To specify an alternate file location, set the environmental variable, SPARK_CONFIG, to the path of the file relative to your project. If you're accustomed to submitting a Spark job with key-values pairs following a --conf flag, these can also be set in a spark-defaults.conf file instead. For a list of valid key-value pairs, refer to *Spark Configuration*.

Administrators can set environment variable paths in the /etc/spark/conf/spark-env.sh file.

Related Information

Spark Configuration

Managing Memory Available for Spark Drivers

By default, the amount of memory allocated to Spark driver processes is set to a 0.8 fraction of the total memory allocated for the runtime container. If you want to allocate more or less memory to the Spark driver process, you can override this default by setting the spark.driver.memory property in spark-defaults.conf (as described above).

Note: The memory allocated to a CML session does not include memory taken by Spark executors.

Managing Dependencies for Spark 2 Jobs

As with any Spark job, you can add external packages to the executor on startup. To add external dependencies to Spark jobs, specify the libraries you want added by using the appropriate configuration parameter in a spark-defaul ts.conf file.

The following table lists the most commonly used configuration parameters for adding dependencies and how they can be used:

Property	Description
spark.files	Comma-separated list of files to be placed in the working directory of each Spark executor.
spark.submit.pyFiles	Comma-separated list of .zip, .egg, or .py files to place on PYTHONPATH for Python applications.
spark.jars	Comma-separated list of local jars to include on the Spark driver and Spark executor classpaths.
spark.jars.packages	Comma-separated list of Maven coordinates of jars to include on the Spark driver and Spark executor classpaths. When configured, Spark will search the local Maven repo, and then Maven central and any additional remote repositories configured by spark.jars.ivy. The format for the coordinates are groupId:artifactId:version.

Property	Description
spark.jars.ivy	Comma-separated list of additional remote repositories to search for the coordinates given with spark.jars.packages.

Example spark-defaults.conf

Here is a sample spark-defaults.conf file that uses some of the Spark configuration parameters discussed in the previous section to add external packages on startup.

```
spark.jars.packages org.scalaj:scalaj-http_2.11:2.3.0
spark.jars my_sample.jar
spark.files data/test_data_1.csv,data/test_data_2.csv
```

spark.jars.packages

The scalaj package will be downloaded from Maven central and included on the Spark driver and executor classpaths.

spark.jars

The pre-existing jar, my_sample.jar, residing in the root of this project will be included on the Spark driver and executor classpaths.

spark.files

The two sample data sets, test_data_1.csv and test_data_2.csv, from the /data directory of this project will be distributed to the working directory of each Spark executor.

For more advanced configuration options, visit the Apache 2 reference documentation.

Related Information

Spark Configuration

LOG4J Configuration

Natural Language Toolkit

Making Python on Apache Hadoop Easier with Anaconda and CDH

Spark Log4j Configuration

Cloudera Machine Learning allows you to update Spark's internal logging configuration on a per-project basis.

Spark 2 uses Apache Log4j, which can be configured through a properties file. By default, a log4j.properties file found in the root of your project will be appended to the existing Spark logging properties for every session and job. To specify a custom location, set the environmental variable LOG4J_CONFIG to the file location relative to your project.

The Log4j documentation has more details on logging options.

Increasing the log level or pushing logs to an alternate location for troublesome jobs can be very helpful for debugging. For example, this is a log4j.properties file in the root of a project that sets the logging level to INFO for Spark jobs.

```
shell.log.level=INFO
```

PySpark logging levels should be set as follows:

```
log4j.logger.org.apache.spark.api.python.PythonGatewayServer=<LOG_LEVEL>
```

And Scala logging levels should be set as:

```
log4j.logger.org.apache.spark.repl.Main=<LOG_LEVEL>
```

Setting Up an HTTP Proxy for Spark 2

If you are using an HTTP proxy, you must set the Spark configuration parameter extraJavaOptions at runtime to be able to support web-related actions in Spark.

```
spark.driver.extraJavaOptions= \
-Dhttp.proxyHost=<YOUR HTTP PROXY HOST> \
-Dhttp.proxyPort=<HTTP PORT> \
-Dhttps.proxyHost=<YOUR HTTPS PROXY HOST> \
-Dhttps.proxyPort=<HTTPS PORT>
```

Spark Web UIs

This topic describes how to access Spark web UIs from the CML UI.

Spark 2 exposes one web UI for each Spark application driver running in Cloudera Machine Learning. The UI will be running within the container, on the port specified by the environmental variable CDSW_SPARK_PORT. By default, CDSW_SPARK_PORT is set to 20049. The web UI will exist only as long as a SparkContext is active within a session. The port is freed up when the SparkContext is shutdown.

Spark 2 web UIs are available in browsers at: https://spark-<$CDSW_ENGINE_ID>.<$CDSW_DOMAIN>. To access the UI while you are in an active session, click the grid icon in the upper right hand corner of the Cloudera Machine Learning web application, and select Spark UI from the dropdown. Alternatively, the Spark UI is also available as a tab in the session itself. For a job, navigate to the job overview page and click the History tab. Click on a job run to open the session output for the job.

Using Spark 2 from Python

Cloudera Machine Learning supports using Spark 2 from Python via PySpark. This topic describes how to set up and test a PySpark project.

PySpark Environment Variables

The default Cloudera Machine Learning engine currently includes Python 2.7.17 and Python 3.6.9. To use PySpark with lambda functions that run within the CDH cluster, the Spark executors must have access to a matching version of Python. For many common operating systems, the default system Python will not match the minor release of Python included in Machine Learning.

To ensure that the Python versions match, Python can either be installed on every CDH host or made available per job run using Spark's ability to distribute dependencies. Given the size of a typical isolated Python environment, Cloudera recommends installing Python 2.7 and 3.6 on the cluster if you are using PySpark with lambda functions.

You can install Python 2.7 and 3.6 on the cluster using any method and set the corresponding PYSPARK_PYTHON environment variable in your project. Cloudera Machine Learning includes a separate environment variable for Python 3 sessions called PYSPARK3_PYTHON. Python 2 sessions continue to use the default PYSPARK_PYTHON variable. This will allow you to run Python 2 and Python 3 sessions in parallel without either variable being overridden by the other.

Creating and Running a PySpark Project

To get started quickly, use the PySpark template project to create a new project. For instructions, see *Create a Project from a Built-in Template*.

To run a PySpark project, navigate to the project's overview page, open the workbench console and launch a Python session. For detailed instructions, see *Native Workbench Console and Editor*.

Testing a PySpark Project in Spark Local Mode

Spark's local mode is often useful for testing and debugging purposes. Use the following sample code snippet to start a PySpark session in local mode.

```
from pyspark.sql import SparkSession

spark = SparkSession\
    .builder \
    .appName("LocalSparkSession") \
    .master("local") \
    .getOrCreate()
```

For more details, refer to the Spark documentation: *Running Spark Application*.

Related Information

Native Workbench Console and Editor

Example: Montecarlo Estimation

Within the template PySpark project, pi.py is a classic example that calculates Pi using the Montecarlo Estimation.

What follows is the full, annotated code sample that can be saved to the pi.py file.

```
# # Estimating $\pi$
#
# This PySpark example shows you how to estimate $\pi$ in parallel
# using Monte Carlo integration.

from __future__ import print_function
import sys
from random import random
from operator import add
# Connect to Spark by creating a Spark session
from pyspark.sql import SparkSession
spark = SparkSession\
    .builder\
    .appName("PythonPi")\
    .getOrCreate()

partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
n = 100000 * partitions

def f(_):
    x = random() * 2 - 1
    y = random() * 2 - 1
    return 1 if x ** 2 + y ** 2 < 1 else 0

# To access the associated SparkContext
count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f)
.reduce(add)
print("Pi is roughly %f" % (4.0 * count / n))

spark.stop()
```

Example: Locating and Adding JARs to Spark 2 Configuration

This example shows how to discover the location of JAR files installed with Spark 2, and add them to the Spark 2 configuration.

```
# # Using Avro data
#
# This example shows how to use a JAR file on the local filesystem on
# Spark on Yarn.

from __future__ import print_function
import os,sys
import os.path
from functools import reduce
from pyspark.sql import SparkSession
from pyspark.files import SparkFiles

# Add the data file to HDFS for consumption by the Spark executors.
!hdfs dfs -put resources/users.avro /tmp

# Find the example JARs provided by the Spark parcel. This parcel
# is available on both the driver, which runs in Cloudera Machine Learning,
 and the
# executors, which run on Yarn.
exampleDir = os.path.join(os.environ["SPARK_HOME"], "examples/jars")
exampleJars = [os.path.join(exampleDir, x) for x in os.listdir(exampleDir)]
# Add the Spark JARs to the Spark configuration to make them available for
use.
spark = SparkSession\
    .builder\
    .config("spark.jars", ",".join(exampleJars))\
    .appName("AvroKeyInputFormat")\
    .getOrCreate()
sc = spark.sparkContext

# Read the schema.
schema = open("resources/user.avsc").read()
conf = {"avro.schema.input.key": schema }
avro_rdd = sc.newAPIHadoopFile(
    "/tmp/users.avro", # This is an HDFS path!
    "org.apache.avro.mapreduce.AvroKeyInputFormat",
    "org.apache.avro.mapred.AvroKey",
    "org.apache.hadoop.io.NullWritable",
    keyConverter="org.apache.spark.examples.pythonconverters.AvroWrapperT
oJavaConverter",
    conf=conf)
output = avro_rdd.map(lambda x: x[0]).collect()
for k in output:
    print(k)
spark.stop()
```

Using Spark 2 from R

R users can access Spark 2 using sparklyr. Although Cloudera does not ship or support sparklyr, we do recommend using sparklyr as the R interface for Cloudera Machine Learning.

Before you begin

The spark_apply() function requires the R Runtime environment to be pre-installed on your cluster. This will likely require intervention from your cluster administrator. For details, refer the RStudio documentation.

Procedure

1. Install the latest version of sparklyr:

```
install.packages("sparklyr")
```

2. Optionally, connect to a local or remote Spark 2 cluster:

```
## Connecting to Spark 2
# Connect to an existing Spark 2 cluster in YARN client mode using the
spark_connect function.
library(sparklyr)
system.time(sc <- spark_connect(master = "yarn-client"))
# The returned Spark 2 connection (sc) provides a remote dplyr data source
 to the Spark 2 cluster.
```

For a complete example, see *Importing Data into Cloudera Machine Learning*.

Related Information

sparklyr: R interface for Apache Spark

sparklyr Requirements

Using Spark 2 from Scala

This topic describes how to set up a Scala project for CDS 2.x Powered by Apache Spark along with a few associated tasks. Cloudera Machine Learning provides an interface to the Spark 2 shell (v 2.0+) that works with Scala 2.11.

Unlike PySpark or Sparklyr, you can access a SparkContext assigned to the *spark* (SparkSession) and *sc* (SparkContext) objects on console startup, just as when using the Spark shell.

By default, the application name will be set to CML_*sessionID*, where sessionId is the id of the session running your Spark code. To customize this, set the spark.app.name property to the desired application name in a spark-defaults.conf file.

Pi.scala is a classic starting point for calculating Pi using the Montecarlo Estimation.

This is the full, annotated code sample.

```
//Calculate pi with Monte Carlo estimation
import scala.math.random
//make a very large unique set of 1 -> n
val partitions = 2
val n = math.min(100000L * partitions, Int.MaxValue).toInt
val xs = 1 until n

//split up n into the number of partitions we can use
val rdd = sc.parallelize(xs, partitions).setName("'N values rdd'")

//generate a random set of points within a 2x2 square
val sample = rdd.map { i =>
  val x = random * 2 - 1
  val y = random * 2 - 1
  (x, y)
}.setName("'Random points rdd'")

//points w/in the square also w/in the center circle of r=1
val inside = sample.filter { case (x, y) => (x * x + y * y < 1) }.setName(
"'Random points inside circle'")
val count = inside.count()

//Area(circle)/Area(square) = inside/n => pi=4*inside/n
```

```
println("Pi is roughly " + 4.0 * count / n)
```

Key points to note:

- import scala.math.random

  Importing included packages works just as in the shell, and need only be done once.
- Spark context (*sc*).

  You can access a SparkContext assigned to the variable *sc* on console startup.

  ```
  val rdd = sc.parallelize(xs, partitions).setName("'N values rdd'")
  ```

Managing Dependencies for Spark 2 and Scala

This topic demonstrates how to manage dependencies on local and external files or packages.

Example: Read Files from the Cluster Local Filesystem

Use the following command in the terminal to read text from the local filesystem. The file must exist on all hosts, and the same path for the driver and executors. In this example you are reading the file ebay-xbox.csv.

```
sc.textFile(“file:///tmp/ebay-xbox.csv”)
```

Adding Remote Packages

External libraries are handled through line magics. Line magics in the Toree kernel are prefixed with %. You can use Apache Toree's AddDeps magic to add dependencies from Maven central. You must specify the company name, artifact ID, and version. To resolve any transitive dependencies, you must explicitly specify the --transitive flag.

```
%AddDeps org.scalaj scalaj-http_2.11 2.3.0
import scalaj.http._
val response: HttpResponse[String] = Http("http://www.omdbapi.com/").param(
"t","crimson tide").asString
response.body
response.code
response.headers
response.cookies
```

Adding Remote or Local Jars

You can use the AddJars magic to distribute local or remote JARs to the kernel and the cluster. Using the *-f* option ignores cached JARs and reloads.

```
%AddJar http://example.com/some_lib.jar -f
%AddJar file:/path/to/some/lib.jar
```

Using GPUs for Cloudera Machine Learning projects

A GPU is a specialized processor that can be used to accelerate highly parallelized computationally-intensive workloads. Because of their computational power, GPUs have been found to be particularly well-suited to deep learning workloads. Ideally, CPUs and GPUs should be used in tandem for data engineering and data science workloads. A typical machine learning workflow involves data preparation, model training, model scoring, and model fitting. You can use existing general-purpose CPUs for each stage of the workflow, and optionally accelerate the math-intensive steps with the selective application of special-purpose GPUs. For example, GPUs allow you to accelerate model fitting using frameworks such as Tensorflow, PyTorch, and Keras.

By enabling GPU support, data scientists can share GPU resources available on Cloudera Machine Learning workspaces. Users can request a specific number of GPU instances, up to the total number available, which are then allocated to the running session or job for the duration of the run.

For information on installing your GPUs, see *CDP Private Cloud Data Services Installation Software Requirements*, below.

Enabling GPUs on ML Workspaces

Note: Nvidia GPU Edition comes with CUDA 11.1 preinstalled.

If you are using a Legacy Engine, to enable GPU usage on Cloudera Machine Learning, select GPUs when you are provisioning the workspace. If your existing workspace does not have GPUs provisioned, contact your ML administrator to provision a new one for you. For instructions, see *Provisioning ML Workspaces*.

Related Information

CDP Private Cloud Experiences Installation Software Requirements
Provision an ML Workspace
Custom CUDA-capable Engine Image
Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment
Project Admins: Enable the CUDA Engine for your Project
Testing GPU Setup
GPU node setup

Using GPUs with Legacy Engines

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

Custom CUDA-capable Engine Image

Note: Before proceeding with creating a custom CUDA-capable engine, the Administrator needs to install the Nvidia plugin.

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

The following sample Dockerfile illustrates an engine on top of which machine learning frameworks such as Tensorflow and PyTorch can be used. This Dockerfile uses a deep learning library from NVIDIA called NVIDIA CUDA Deep Neural Network (cuDNN). For detailed information about compatibility between NVIDIA driver versions and CUDA, refer the cuDNN installation guide (prerequisites).

When creating the Dockerfile for the custom image, you must delete the Cloudera repository that is inaccessible because of the paywall by running the following:

```
RUN rm /etc/apt/sources.list.d/*
```

Make sure you also check with the machine learning framework that you intend to use in order to know which version of cuDNN is needed. As an example, Tensorflow's NVIDIA hardware and software requirements for GPU support are listed in the Tensorflow documentation here. Additionally, the Tensorflow version compatibility matrix for CUDA and cuDNN is documented here.

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

cuda.Dockerfile

```
FROM docker.repository.cloudera.com/cloudera/cdsw/engine:14-cml-2021.05-1

RUN rm /etc/apt/sources.list.d/*
```

```
RUN apt-get update && apt-get install -y --no-install-recommends \
gnupg2 curl ca-certificates && \
curl -fsSL https://developer.download.nvidia.com/compute/cuda/repos/ubuntu18
04/x86_64/7fa2af80.pub | apt-key add - && \
echo "deb https://developer.download.nvidia.com/compute/cuda/repos/ubuntu180
4/x86_64 /" > /etc/apt/sources.list.d/cuda.list && \
echo "deb https://developer.download.nvidia.com/compute/machine-learning/re
pos/ubuntu1804/x86_64 /" > /etc/apt/sources.list.d/nvidia-ml.list && \
apt-get purge --autoremove -y curl && \
rm -rf /var/lib/apt/lists/*

ENV CUDA_VERSION 10.1.243
LABEL com.nvidia.cuda.version="${CUDA_VERSION}"

ENV CUDA_PKG_VERSION 10-1=$CUDA_VERSION-1
RUN apt-get update && apt-get install -y --no-install-recommends \
cuda-cudart-$CUDA_PKG_VERSION && \
cuda-libraries-$CUDA_PKG_VERSION && \
ln -s cuda-10.1 /usr/local/cuda && \
rm -rf /var/lib/apt/lists/*

RUN echo "/usr/local/cuda/lib64" >> /etc/ld.so.conf.d/cuda.conf && \
ldconfig

RUN echo "/usr/local/nvidia/lib" >> /etc/ld.so.conf.d/nvidia.conf && \
echo "/usr/local/nvidia/lib64" >> /etc/ld.so.conf.d/nvidia.conf

ENV PATH /usr/local/nvidia/bin:/usr/local/cuda/bin:${PATH}
ENV LD_LIBRARY_PATH /usr/local/nvidia/lib:/usr/local/nvidia/lib64:/usr/loca
l/cuda-10.2/targets/x86_64-linux/lib/

RUN echo "deb http://developer.download.nvidia.com/compute/machine-learning/
repos/ubuntu1604/x86_64 /" > /etc/apt/sources.list.d/nvidia-ml.list

ENV CUDNN_VERSION 7.6.5.32
LABEL com.nvidia.cudnn.version="${CUDNN_VERSION}"

RUN apt-get update && apt-get install -y --no-install-recommends \
libcudnn7=$CUDNN_VERSION-1+cuda10.1 && \
apt-mark hold libcudnn7 && \
rm -rf /var/lib/apt/lists/*
```

Use the following example command to build the custom engine image using the cuda.Dockerfile command:

```
docker build --network host -t <company-registry>/CML-cuda:13 . -f cuda.Dock
erfile
```

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

```
docker push <company-registry>/CML-cuda:13
```

Site Admins: Add the Custom CUDA Engine to your Cloudera Machine Learning Deployment

After you create a custom CUDA-capable engine image, you must add the new engine to Cloudera Machine Learning.

About this task

You must have the Site Administrator role to perform this task.

Procedure

1. Sign in to Cloudera Machine Learning.
2. Click Admin.
3. Go to the Engines tab.
4. Under Engine Images, add the custom CUDA-capable engine image created in the previous step.

   This allows project administrators across the deployment to start using this engine in their jobs and sessions.
5. Site administrators can also set a limit on the maximum number of GPUs that can be allocated per session or job. From the Maximum GPUs per Session/Job dropdown, select the maximum number of GPUs that can be used by an engine.
6. Click Update.

Project Admins: Enable the CUDA Engine for your Project

You can make the CUDA-capable engine the default engine for workloads within a particular project.

Before you begin

You must be a Project administrator to specify the default engine used for workloads within a particular project.

Procedure

1. Navigate to your project's Overview page.
2. Click Settings.
3. Go to the Engines tab.
4. Under Engine Image, select the CUDA-capable engine image from the dropdown.

Testing GPU Setup

Use these code samples to test that your GPU setup works with several common deep learning libraries. The specific versions of libraries depend on the particular GPU used and the GPU driver version. You can use this testing for GPU setup using Legacy Engines.

1. Go to a project that is using the CUDA engine and click Open Workbench.
2. Launch a new session with GPUs.
3. Run the following command in the workbench command prompt to verify that the driver was installed correctly:

   ```
   ! /usr/bin/nvidia-smi
   ```
4. Use any of the following code samples to confirm that the new engine works with common deep learning libraries.

PyTorch

```
!pip3 install torch==1.4.0
from torch import cuda
assert cuda.is_available()
assert cuda.device_count() > 0
print(cuda.get_device_name(cuda.current_device()))
```

Note: The PyTorch installation requires at least 4 GB of memory.

Tensorflow

```
!pip3 install tensorflow-gpu==2.1.0
from tensorflow.python.client import device_lib
```

```
assert 'GPU' in str(device_lib.list_local_devices())
device_lib.list_local_devices()
```

Keras

```
!pip3 install keras
from keras import backend
assert len(backend.tensorflow_backend._get_available_gpus()) > 0
print(backend.tensorflow_backend._get_available_gpus())
```

Running an Experiment (QuickStart)

This topic walks you through a simple example to help you get started with experiments in Cloudera Machine Learning.

The following steps describe how to launch an experiment from the Workbench console. In this example we are going to run a simple script that adds all the numbers passed as arguments to the experiment.

1. Go to the project Overview page.
2. Click Open Workbench.
3. Create/modify any project code as needed. You can also launch a session to simultaneously test code changes on the interactive console as you launch new experiments.

   As an example, you can run this Python script that accepts a series of numbers as command-line arguments and prints their sum.

   add.py

   ```
   import sys
   import cdsw

   args = len(sys.argv) - 1
   sum = 0
   x = 1

   while (args >= x):
       print ("Argument %i: %s" % (x, sys.argv[x]))
       sum = sum + int(sys.argv[x])
       x = x + 1

   print ("Sum of the numbers is: %i." % sum)
   ```

   To test the script, launch a Python session and run the following command from the workbench command prompt:

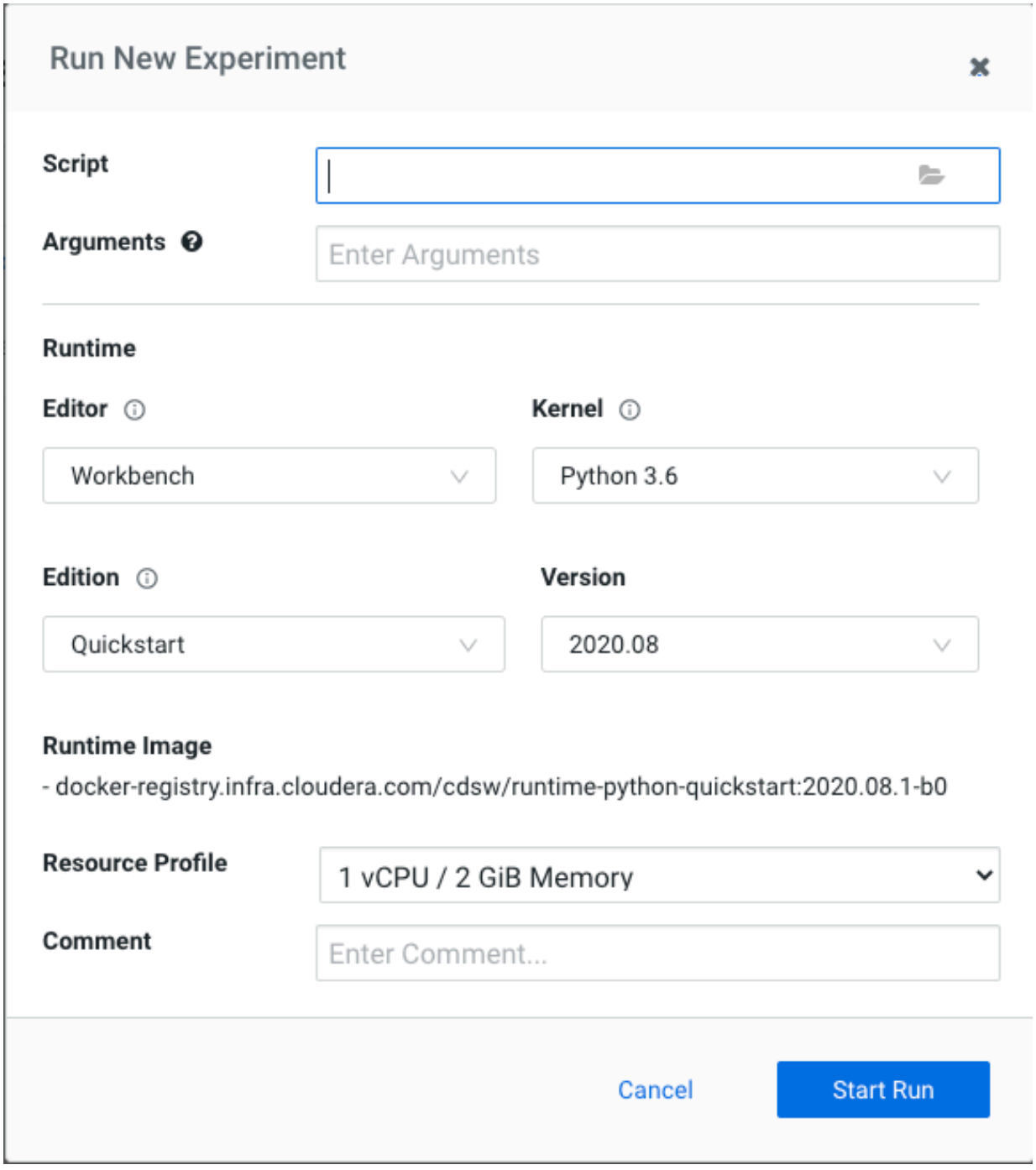
   ```
   !python add.py 1 2 3 4
   ```

4. Click Run Experiment. If you're already in an active session, click Run Run Experiment . Fill out the following fields:
   - Script - Select the file that will be executed for this experiment.
   - Arguments - If your script requires any command line arguments, enter them here.

     Note: Arguments are not supported with Scala experiments.

   - Engine Kernel and Resource Profile - Select the kernel and computing resources needed for this experiment.

   For this example we will run the add.py script and pass some numbers as arguments.



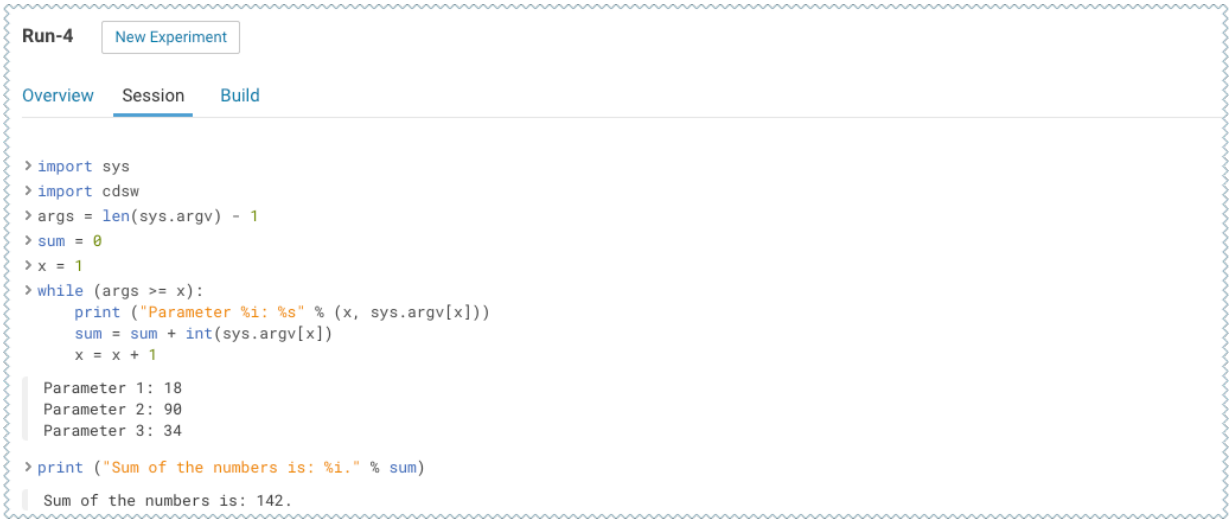
5. Click Start Run.

6. To track progress for the run, go back to the project Overview. On the left navigation bar click Experiments. You should see the experiment you've just run at the top of the list. Click on the Run ID to view an overview for each individual run. Then click Build.

   On this Build tab you can see realtime progress as Cloudera Machine Learning builds the Docker image for this experiment. This allows you to debug any errors that might occur during the build stage.

7. Once the Docker image is ready, the run will begin execution. You can track progress for this stage by going to the Session tab.

   For example, the Session pane output from running add.py is:



8. (Optional) The cdsw library that is bundled with Cloudera Machine Learning includes some built-in functions that you can use to compare experiments and save any files from your experiments.

   For example, to track the sum for each run, add the following line to the end of the add.py script.

   ```
   cdsw.track_metric("Sum", sum)
   ```

   This will be tracked in the Experiments table:

Related Information

Tracking Metrics

Saving Files

Limitations

This topic lists some of the known issues and limitations associated with experiments.

- Experiments do not store snapshots of project files. You cannot automatically restore code that was run as part of an experiment.
- Experiments will fail if your project filesystem is too large for the Git snapshot process. As a general rule, any project files (code, generated model artifacts, dependencies, etc.) larger than 50 MB must be part of your project's .gitignore file so that they are not included in snapshots for experiment builds.
- Experiments cannot be deleted. As a result, be conscious of how you use the track_metrics and track_file functions.
  - Do not track files larger than 50MB.
  - Do not track more than 100 metrics per experiment. Excessive metric calls from an experiment may cause Cloudera Machine Learning to stop responding.
- The Experiments table will allow you to display only three metrics at a time. You can select which metrics are displayed from the metrics dropdown. If you are tracking a large number of metrics (100 or more), you might notice some performance lag in the UI.
- Arguments are not supported with Scala experiments.
- The track_metrics and track_file functions are not supported with Scala experiments.
- The UI does not display a confirmation when you start an experiment or any alerts when experiments fail.

Related Information

Engines for Experiments and Models

Tracking Metrics

This topic teaches you how to use the track_metric function to log metrics associated with experiments.

The cdsw library includes a track_metric function that can be used to log up to 50 metrics associated with a run, thus allowing accuracy and scores to be tracked over time.

The function accepts input in the form of key value pairs.

```
cdsw.track_metric(key, value)
```

Python

```
cdsw.track_metric(“R_squared”, 0.79)
```

R

```
cdsw::track.metric(“R_squared”, 0.62)
```

These metrics will be available on the project's Experiments tab where you can view, sort, and filter experiments on the values. The table on the Experiments page will allow you to display only three metrics at a time. You can select which metrics are displayed from the metrics dropdown.

Note: This function is not supported with Scala experiments.

Saving Files

This topic teaches you how to use the track_file function to save files associated with experiments.

Cloudera Machine Learning allows you to select which artifacts you'd like to access and evaluate after an experiment is complete. These artifacts could be anything from a text file to an image or a model that you have built through the run.

The cdsw library includes a track_file function that can be used to specify which artifacts should be retained after the experiment is complete.

Python

```
cdsw.track_file('model.pkl')
```

R

```
cdsw::track.file('model.pkl')
```

Specified artifacts can be accessed from the run's Overview page. These files can also be saved to the top-level project filesystem and downloaded from there.

Note: This function is not supported with Scala experiments.

Debugging Issues with Experiments

This topic lists some common issues to watch out for during an experiment's build and execution process:

Experiment spends too long in Scheduling/Built stage

If your experiments are spending too long in any particular stage, check the resource consumption statistics for the cluster. When the cluster starts to run out of resources, often experiments (and other entities like jobs, models) will spend too long in the queue before they can be executed.

Resource consumption by experiments (and jobs, sessions) can be tracked by site administrators on the Admin Activity page.

Experiment fails in the Build stage

During the build stage Cloudera Machine Learning creates a new Docker image for the experiment. You can track progress for this stage on each experiment's Build page. The build logs on this page should help point you in the right direction.

Common issues that might cause failures at this stage include:

- Lack of execute permissions on the build script itself.
- Inability to reach the Python package index or R mirror when installing packages.
- Typo in the name of the build script (cdsw-build.sh). Note that the build process will only run a script called cdsw-build.sh; not any other bash scripts from your project.
- Using pip3 to install packages in cdsw-build.sh, but selecting a Python 2 kernel when you actually launch the experiment. Or vice versa.

Experiment fails in the Execute stage

Each experiment includes a Session page where you can track the output of the experiment as it executes. This is similar to the output you would see if you test the experiment in the workbench console. Any runtime errors will display on the Session page just as they would in an interactive session.

Related Information

Engines for Experiments and Models

Model Training and Deployment Overview

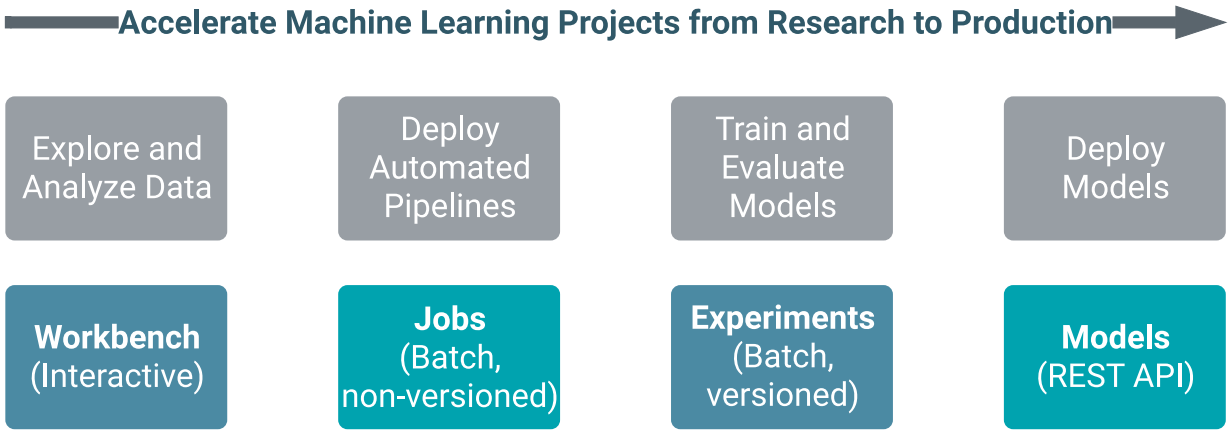
This section provides an overview of model training and deployment using Cloudera Machine Learning.

Machine learning is a discipline that uses computer algorithms to extract useful knowledge from data. There are many different types of machine learning algorithms, and each one works differently. In general however, machine learning algorithms begin with an initial hypothetical model, determine how well this model fits a set of data, and then work on improving the model iteratively. This training process continues until the algorithm can find no additional improvements, or until the user stops the process.

A typical machine learning project will include the following high-level steps that will transform a loose data hypothesis into a model that serves predictions.

1. Explore and experiment with and display findings of data
2. Deploy automated pipelines of analytics workloads
3. Train and evaluate models
4. Deploy models as REST APIs to serve predictions

With Cloudera Machine Learning, you can deploy the complete lifecycle of a machine learning project from research to deployment.



Experiments

This topic introduces you to experiments, and the challenge this feature aims to solve.

Cloudera Machine Learning allows data scientists to run batch experiments that track different versions of code, input parameters, and output (both metrics and files).

Challenge

As data scientists iteratively develop models, they often experiment with datasets, features, libraries, algorithms, and parameters. Even small changes can significantly impact the resulting model. This means data scientists need the ability to iterate and repeat similar experiments in parallel and on demand, as they rely on differences in output and scores to tune parameters until they obtain the best fit for the problem at hand. Such a training workflow requires versioning of the file system, input parameters, and output of each training run.

Without versioned experiments you would need intense process rigor to consistently track training artifacts (data, parameters, code, etc.), and even then it might be impossible to reproduce and explain a given result. This can lead to wasted time and effort during collaboration, not to mention the compliance risks introduced.

Solution

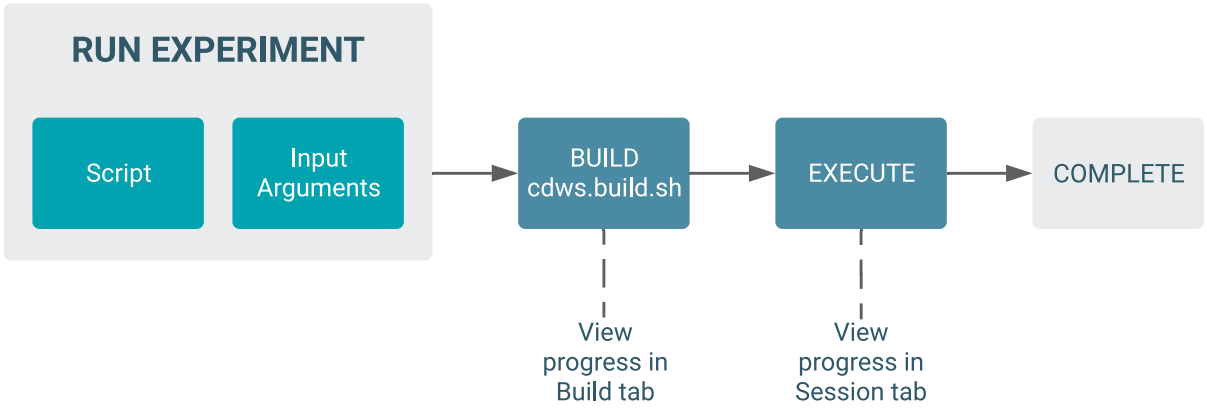
Cloudera Machine Learning uses experiments to facilitate ad-hoc batch execution and model training. Experiments are batch executed workloads where the code, input parameters, and output artifacts are versioned. This feature also provides a lightweight ability to track output data, including files, metrics, and metadata for comparison.

Experiments - Concepts and Terminology

This topic walks you through some basic concepts and terminology related to experiments.

The term experiment refers to a non interactive batch execution script that is versioned across input parameters, project files, and output. Batch experiments are associated with a specific project (much like sessions or jobs) and have no notion of scheduling; they run at creation time. To support versioning of the project files and retain run-level artifacts and metadata, each experiment is executed in an isolated container.

Lifecycle of an Experiment



The rest of this section describes the different stages in the lifecycle of an experiment - from launch to completion.

1. **Launch Experiment**

   In this step you will select a script from your project that will be run as part of the experiment, and the resources (memory/GPU) needed to run the experiment. The engine kernel will be selected by default based on your script. For detailed instructions on how to launch an experiment, see *Getting Started with Cloudera Machine Learning*.

2. **Build**

   When you launch the experiment, Cloudera Machine Learning first builds a new versioned engine image where the experiment will be executed in isolation. This new engine includes:

   - the base engine image used by the project (check Project Settings)
   - a snapshot of the project filesystem
   - environmental variables inherited from the project.
   - packages explicitly specified in the project's build script (cdsw-build.sh)

     It is your responsibility to provide the complete list of dependencies required for the experiment via the cdsw-build.sh file. As part of the engine's build process, Cloudera Machine Learning will run the cdsw-build.sh script and install the packages or libraries requested there on the new image.

   For details about the build process and examples on how to specify dependencies, see Engines for Experiments and Models. .

3. **Schedule**

   Once the engine is built the experiment is scheduled for execution like any other job or session. Once the requested CPU/GPU and memory have been allocated to the experiment, it will move on to the execution stage.

   Note that if your deployment is running low on memory and CPU, your runs may spend some time in this stage.

4. **Execute**

   This is the stage where the script you have selected will be run in the newly built engine environment. This is the same output you would see if you had executed the script in a session in the Workbench console.

   You can watch the execution in progress in the individual run's Session tab.

   You can also go to the project Overview Experiments page to see a table of all the experiments launched within that project and their current status.

   Run ID: A numeric ID that tracks all experiments launched on a Cloudera Machine Learning deployment. It is not limited to the scope of a single user or project.

Related Information

Running an Experiment with Cloudera Machine Learning

Models

Cloudera Machine Learning allows data scientists to build, deploy, and manage models as REST APIs to serve predictions.

Challenge

Data scientists often develop models using a variety of Python/R open source packages. The challenge lies in actually exposing those models to stakeholders who can test the model. In most organizations, the model deployment process will require assistance from a separate DevOps team who likely have their own policies about deploying new code.

For example, a model that has been developed in Python by data scientists might be rebuilt in another language by the devops team before it is actually deployed. This process can be slow and error-prone. It can take months to deploy new models, if at all. This also introduces compliance risks when you take into account the fact that the new re-developed model might not be even be an accurate reproduction of the original model.

Once a model has been deployed, you then need to ensure that the devops team has a way to rollback the model to a previous version if needed. This means the data science team also needs a reliable way to retain history of the models they build and ensure that they can rebuild a specific version if needed. At any time, data scientists (or any other stakeholders) must have a way to accurately identify which version of a model is/was deployed.

Solution

Cloudera Machine Learning allows data scientists to build and deploy their own models as REST APIs. Data scientists can now select a Python or R function within a project file, and Cloudera Machine Learning will:

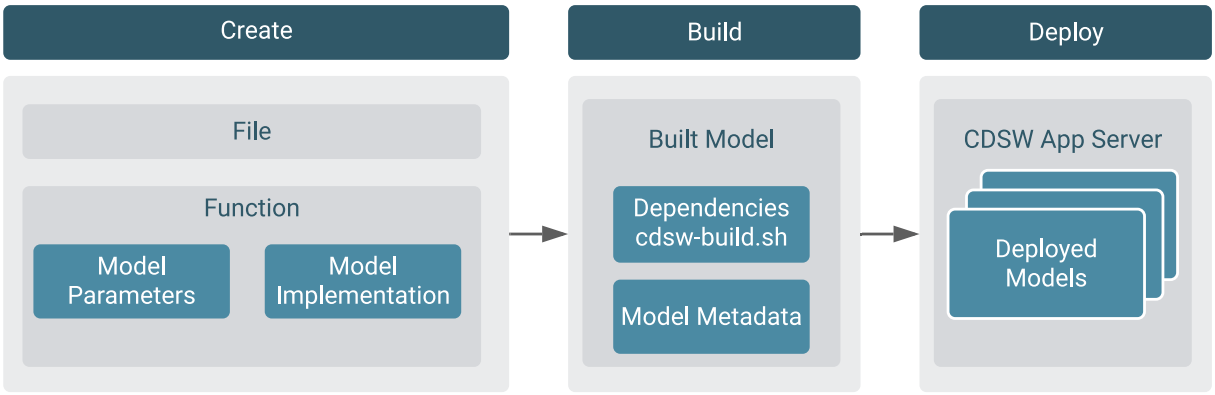
- Create a snapshot of model code, model parameters, and dependencies.
- Package a trained model into an immutable artifact and provide basic serving code.
- Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function’s return type.
- Save the model along with some metadata.
- Deploy a specified number of model API replicas, automatically load balanced.

Models - Concepts and Terminology

Model

> Model is a high level abstract term that is used to describe several possible incarnations of objects created during the model deployment process. For the purpose of this discussion you should note that 'model' does not always refer to a specific artifact. More precise terms (as defined later in this section) should be used whenever possible.

Stages of the Model Deployment Process



The rest of this section contains supplemental information that describes the model deployment process in detail.

Create

- File - The R or Python file containing the function to be invoked when the model is started.
- Function - The function to be invoked inside the file. This function should take a single JSON-encoded object (for example, a python dictionary) as input and return a JSON-encodable object as output to ensure compatibility with any application accessing the model using the API. JSON decoding and encoding for model input/output is built into Cloudera Machine Learning.

  The function will likely include the following components:

  - Model Implementation

    The code for implementing the model (e.g. decision trees, k-means). This might originate with the data scientist or might be provided by the engineering team. This code implements the model's predict function, along with any setup and teardown that may be required.
  - Model Parameters

    A set of parameters obtained as a result of model training/fitting (using experiments). For example, a specific decision tree or the specific centroids of a k-means clustering, to be used to make a prediction.

Build

This stage takes as input the file that calls the function and returns an artifact that implements a single concrete model, referred to as a model build.

- Built Model

  A built model is a static, immutable artifact that includes the model implementation, its parameters, any runtime dependencies, and its metadata. If any of these components need to be changed, for example, code changes to the implementation or its parameters need to be retrained, a new build must be created for the model. Model builds are versioned using build numbers.

  To create the model build, Cloudera Machine Learning creates a Docker image based on the engine designated as the project's default engine. This image provides an isolated environment where the model implementation code will run.

  To configure the image environment, you can specify a list of dependencies to be installed in a build script called cdsw-build.sh.

  For details about the build process and examples on how to install dependencies, see *Engines for Experiments and Models*.

- Build Number:

  Build numbers are used to track different versions of builds within the scope of a single model. They start at 1 and are incremented with each new build created for the model.

Deploy

This stage takes as input the memory/CPU resources required to power the model, the number of replicas needed, and deploys the model build created in the previous stage to a REST API.

- Deployed Model

  A deployed model is a model build in execution. A built model is deployed in a model serving environment, likely with multiple replicas.
- Environmental Variable

  You can set environmental variables each time you deploy a model. Note that models also inherit any environment variables set at the project and global level. (For more information see *Engine Environment Variables*.) However, in case of any conflicts, variables set per-model will take precedence.

  Note: If you are using any model-specific environmental variables, these must be specified every time you re-deploy a model. Models do not inherit environmental variables from previous deployments.
- Model Replicas

  The engines that serve incoming requests to the model. Note that each replica can only process one request at a time. Multiple replicas are essential for load-balancing, fault tolerance, and serving concurrent requests. Cloudera Machine Learning allows you to deploy a maximum of 9 replicas per model.
- Deployment ID

  Deployment IDs are numeric IDs used to track models deployed across Cloudera Machine Learning. They are not bound to a model or project.

Related Information

Experiments - Concepts and Terminology

Engines for Experiments and Models

Engines Environment Variables

Challenges with Machine Learning in production

One of the hardest parts of Machine Learning (ML) is deploying and operating ML models in production applications. These challenges fall maily into the following categories: model deployment and serving, model monitoring, and model governance.

Challenges with model deployment and serving

After models are trained and ready to deploy in a production environment, lack of consistency with model deployment and serving workflows can present challenges in terms of scaling your model deployments to meet the increasing numbers of ML usecases across your business.

Many model serving and deployment workflows have repeatable, boilerplate aspects which you can automate using modern DevOps techniques like high frequency deployment and microservices architectures. This approach can enable the ML engineers to focus on the model instead of the surrounding code and infrastructure.

Challenges with model monitoring

Machine Learning (ML) models predict the world around them which is constantly changing. The unique and complex nature of model behavior and model lifecycle present challenges after the models are deployed.

Cloudera Machine Learning provides you the capability to monitor the performance of the model on two levels: technical performance (latency, throughput, and so on similar to an Application Performance Management), and mathematical performance (is the model predicting correctly, is the model biased, and so on).

There are two types of metrics that are collected from the models:

- Time series metrics: Metrics measured in-line with model prediction. It can be useful to track the changes in these values over time. It is the finest granular data for the most recent measurement. To improve performance, older data is aggregated to reduce data records and storage.
- Post-prediction metrics: Metrics that are calculated after prediction time, based on ground truth and/or batches (aggregates) of time series metrics. To collect metrics from the models, the Python SDK has been extended to include the following functions that you can use to store different types of metrics:

To collect metrics from the models, the Python SDK has been extended to include the following functions that you can use to store different types of metrics:

- track_metrics: Tracks the metrics generated by experiments and models.
- read_metrics: Reads the metrics already tracked for a deployed model, within a given window of time.
- track_delayed_metrics: Tracks metrics that correspond to individual predictions, but aren't known at the time the prediction is made. The most common instances are ground truth and metrics derived from ground truth such as error metrics.
- track_aggregate_metrics: Registers metrics that are not associated with any particular prediction. This function can be used to track metrics accumulated and/or calculated over a longer period of time.

The following two use-cases show how you can use these functions:

- Tracking accuracy of a model over time
- Tracking drift

Usecase 1: Tracking accuracy of a model over time

Consider the case of a large telco. When a customer service representative takes a call from a customer, a web application presents an estimate of the risk that the customer will churn. The service representative takes this risk into account when evaluating whether to offer promotions.

The web application obtains the risk of churn by calling into a model hosted on Cloudera Machine Learning (CML). For each prediction thus obtained, the web application records the UUID into a datastore alongside the customer ID. The prediction itself is tracked in CML using the track_metrics function.

At some point in the future, some customers do in fact churn. When a customer churns, they or another customer service representative close their account in a web application. That web application records the churn event, which is ground truth for this example, in a datastore.

An ML engineer who works at the telco wants to continuously evaluate the suitability of the risk model. To do this, they create a recurring CML job. At each run, the job uses the read_metrics function to read all the predictions that were tracked in the last interval. It also reads in recent churn events from the ground truth datastore. It joins the churn events to the predictions and customer ID's using the recorded UUID's, and computes an Receiver operating characteristic (ROC) metric for the risk model. The ROC is tracked in the metrics store using the track_aggregate_ metrics function.

Note: You can store the ground truth in an external datastore, such as Cloudera Data Warehouse or in the metrics store.

Use-case 2: Tracking drift

Instead of or in addition to computing ROC, the ML engineer may need to track various types of drift. Drift metrics are especially useful in cases where ground truth is unavailable or is difficult to obtain.

The definition of drift is broad and somewhat nebulous and practical approaches to handling it are evolving, but drift is always about changing distributions. The distribution of the input data seen by the model may change over time and deviate from the distribution in the training dataset, and/or the distribution of the output variable may change, and/or the relationship between input and output may change.

All drift metrics are computed by aggregating batches of predictions in some way. As in the use case above, batches of predictions can be read into recurring jobs using the read_metrics function, and the drift metrics computed by the job can be tracked using the track_aggregate_metrics function.

Challenges with model governance

Businesses implement ML models across their entire organization, spanning a large spectrum of usecases. When you start deploying more than just a couple models in production, a lot of complex governance and management challenges arise.

Almost all the governance needs for ML are associated with data and are tied directly to the data management practice in your organization. For example, what data can be used for certain applications, who should be able to access what data, and based on what data are models created.

Some of the other unique governance challenges that you could encounter are:

- How to gain visibility into the impact your models have on your customers?
- How can you ensure you are still compliant with both governmental and internal regulations?
- How does your organization’s security practices apply to the models in production?

Ultimately, the needs for ML governance can be distilled into the following key areas: model visibility, and model explainability, interpretability, and reproducibility.

Model visibility

A basic requirement for model governance is enabling teams to understand how machine learning is being applied in their organizations. This requires a canonical catalog of models in use. In the absence of such a catalog, many organizations are unaware of how their models work, where they are deployed, what they are being used for, and so on. This leads to repeated work, model inconsistencies, recomputing features, and other inefficiencies.

Model explainability, interpretability, and reproducibility

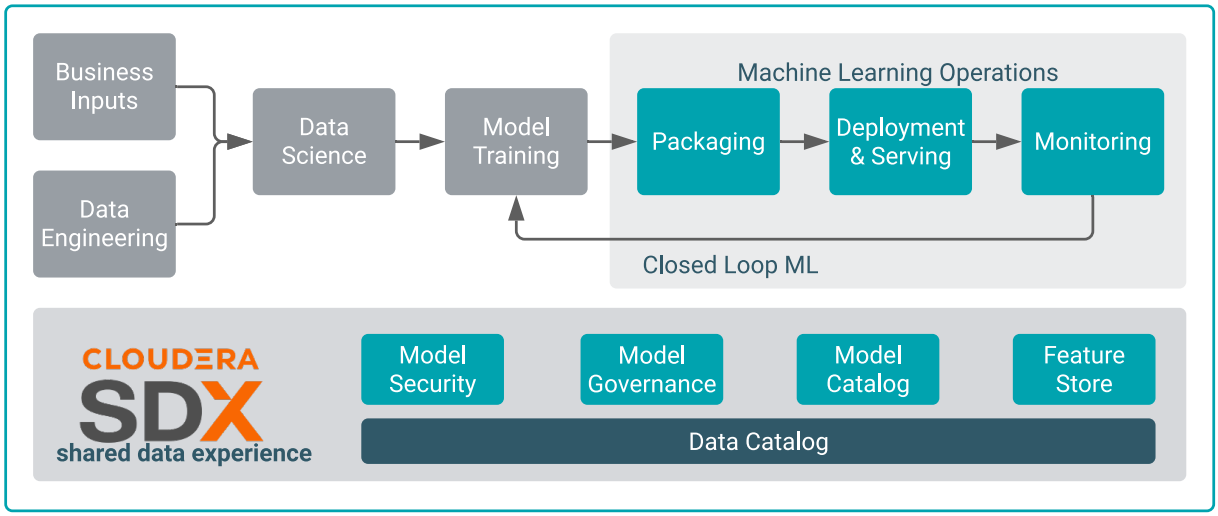
Models are often seen as a black box: data goes in, something happens, and a prediction comes out. This lack of transparency is challenging on a number of levels and is often represented in loosely related terms explainability, interpretability, and reproducibility.

- Explainability: Indicates the description of the internal mechanics of an Machine Learning (ML) model in human terms
- Interpretability: Indicates the ability to:
  - Understand the relationship between model inputs, features and outputs
  - Predict the response to changes in inputs
- Reproducibility: Indicates the ability to reproduce the output of a model in a consistent fashion for the same inputs

To solve these challenges, CML provides an end-to-end model governance and monitoring workflow that gives organizations increased visibility into their machine learning workflows and aims to eliminate the blackbox nature of most machine learning models.

The following image shows the end-to-end producltion ML workflow:

Figure 10: Production ML Workflow



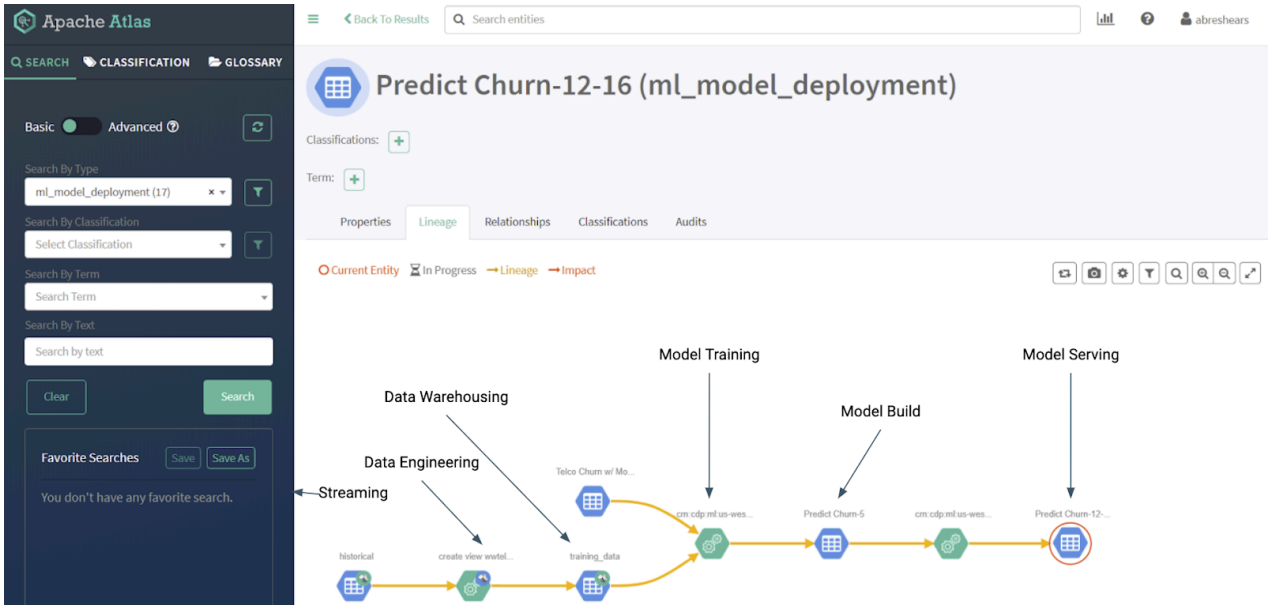
Model governance using Apache Atlas

To address governance challenges, Cloudera Machile Learning uses Apache Atlas to automatically collect and visualize lineage information for data used in Machine Learning (ML) workflows — from training data to model deployments.

Lineage is a visual representation of the project. The lineage information includes visualization of the relationships between model entities such as code, model builds, deployments, and so on. and the processes that carry out transformations on the data involved, such as create project, build model, deploy model, and so on.

The Apache Atlas type system has pre-built governance features that can be used to define ML metadata objects. A type in Atlas is a definition of the metadata stored to describe a data asset or other object or process in an environment. For ML governance, Cloudera has designed new Atlas types that capture ML entities and processes as Atlas metadata objects.

In addition to the definition of the types, Atlas also captures the relationship between the entities and processes to visualize the end-to-end lineage flow, as shown in the following image. The blue hexagons represent an entity (also called the noun) and the green hexagons represent a process (also called the verb).



The ML metadata definition closely follows the actual machine learning workflow. Training data sets are the starting point for a model lineage flow. These data sets can be tables from a data warehouse or an embedded csv file. Once a data set has been identified, the lineage follows into training, building and deploying the model.

See *ML operations entities created in Atlas* for a list of metadata that Atlas collects from each CML workspace. Metadata is collected from machine learning projects, model builds, and model deployments, and the processes that create these entities.

Creating and Deploying a Model

This topic describes a simple example of how to create and deploy a model using Cloudera Machine Learning.

Using Cloudera Machine Learning, you can create any function within a script and deploy it to a REST API. In a machine learning project, this will typically be a predict function that will accept an input and return a prediction based on the model's parameters.

For the purpose of this quick start demo we are going to create a very simple function that adds two numbers and deploy it as a model that returns the sum of the numbers. This function will accept two numbers in JSON format as input and return the sum.

For CML UI

1. Create a new project. Note that models are always created within the context of a project.
2. Click New Session and launch a new Python 3 session.
3. Create a new file within the project called add_numbers.py. This is the file where we define the function that will be called when the model is run. For example:

   add_numbers.py

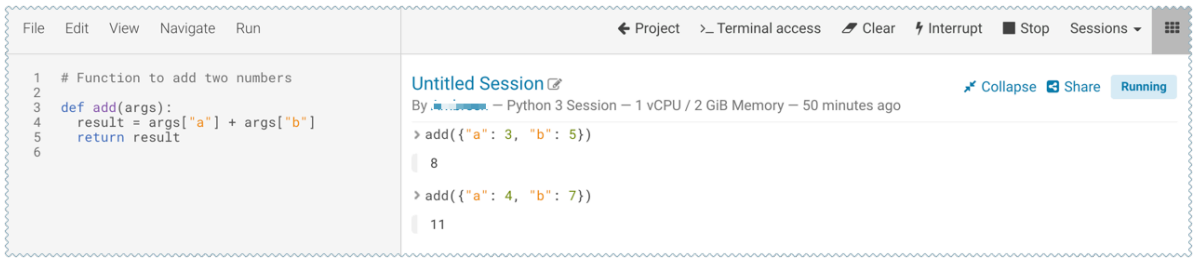
   ```
   def add(args):
     result = args["a"] + args["b"]
   ```

```
    return result
```

Note: In practice, do not assume that users calling the model will provide input in the correct format or enter good values. Always perform input validation.

4. Before deploying the model, test it by running the add_numbers.py script, and then calling the add function directly from the interactive workbench session. For example:

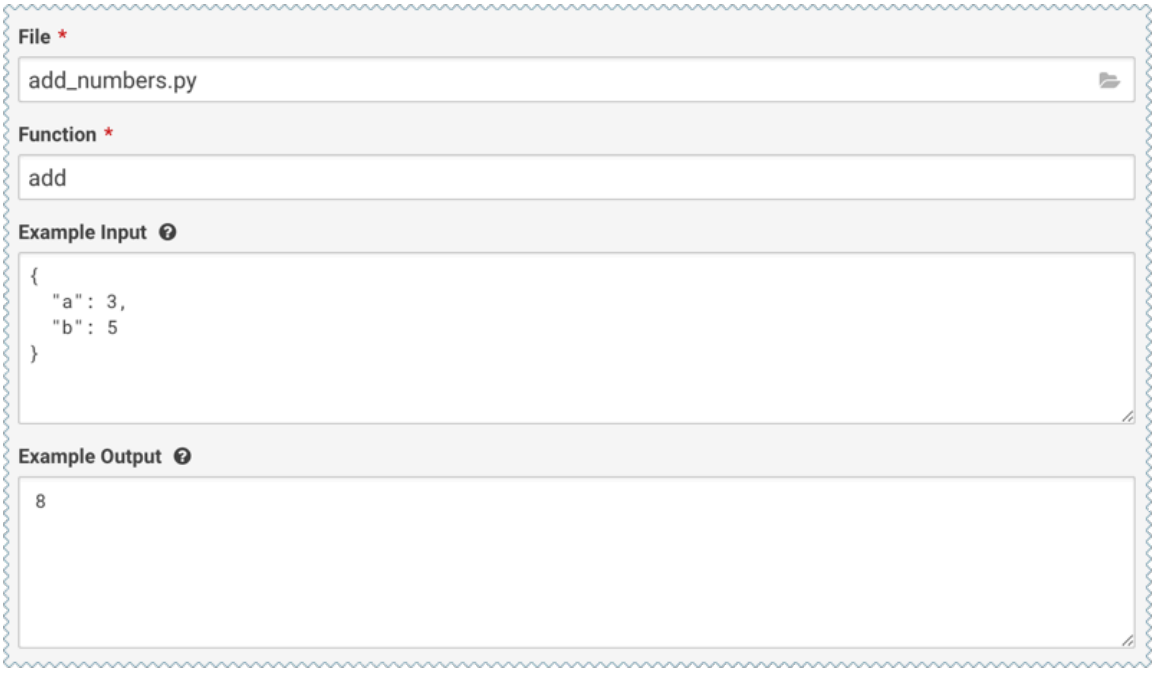
```
add({"a": 3, "b": 5})
```



5. Deploy the add function to a REST endpoint.

   a. Go to the project Overview page.
   b. Click Models New Model .
   c. Give the model a Name and Description.
   d. Enter details about the model that you want to build. In this case:

   - File: add_numbers.py
   - Function: add
   - Example Input: {"a": 3, "b": 5}
   - Example Output: 8



   e. Select the resources needed to run this model, including any replicas for load balancing.

   f. Click Deploy Model.

6. Click on the model to go to its Overview page. Click Builds to track realtime progress as the model is built and deployed. This process essentially creates a Docker container where the model will live and serve requests.

7. Once the model has been deployed, go back to the model Overview page and use the Test Model widget to make sure the model works as expected.

   If you entered example input when creating the model, the Input field will be pre-populated with those values. Click Test. The result returned includes the output response from the model, as well as the ID of the replica that served the request.

   Model response times depend largely on your model code. That is, how long it takes the model function to perform the computation needed to return a prediction. It is worth noting that model replicas can only process one request at a time. Concurrent requests will be queued until the model can process them.

For CML APIv2

To create and deploy a model using the API, follow this example:

This example demonstrates the use of the Models API. To run this example, first do the following:

1. Create a project with the Python template and a legacy engine.
2. Start a session.
3. Run !pip3 install sklearn
4. Run fit.py

The example script first obtains the project ID, then creates and deploys a model.

```
projects = client.list_projects(search_filter=json.dumps({"name": “<your
 project name>”}))
project = projects.projects[0] # assuming only one project is returned by
 the above query
model_body = cmlapi.CreateModelRequest(project_id=project.id, name="Demo
 Model", description="A simple model")
model = client.create_model(model_body, project.id)
model_build_body = cmlapi.CreateModelBuildRequest(project_id=project.id,
 model_id=model.id, file_path="predict.py", function_name="predict", ker
nel="python3")
model_build = client.create_model_build(model_build_body, project.id, mod
el.id)
while model_build.status not in [“built”, “build failed”]:
 print(“waiting for model to build...”)
time.sleep(10)
 model_build = client.get_model_build(project.id, model.id, model_build
.id)
if model_build.status == “build failed”:
```

```
 print("model build failed, see UI for more information")
 sys.exit(1)
print("model built successfully!")
model_deployment_body = cmlapi.CreateModelDeploymentRequest(project_id=p
roject.id, model_id=model.id, build_id=model_build.id)
model_deployment = client.create_model_deployment(model_deployment_body,
project.id, model.id, build.id)
while model_deployment.status not in ["stopped", "failed", "deployed"]:
 print("waiting for model to deploy...")
 time.sleep(10)
 model_deployment = client.get_model_deployment(project.id, model.id, m
odel_build.id, model_deployment.id)
if model_deployment.status != "deployed":
 print("model deployment failed, see UI for more information")
 sys.exit(1)
print("model deployed successfully!")
```

Usage Guidelines

This section calls out some important guidelines you should keep in mind when you start deploying models with Cloudera Machine Learning.

Model Code

Models in Cloudera Machine Learning are designed to run any code that is wrapped into a function. This means you can potentially deploy a model that returns the result of a SELECT * query on a very large table. However, Cloudera strongly recommends against using the models feature for such use cases.

As a best practice, your models should be returning simple JSON responses in near-real time speeds (within a fraction of a second). If you have a long-running operation that requires extensive computing and takes more than 15 seconds to complete, consider using batch jobs instead.

Model Artifacts

Once you start building larger models, make sure you are storing these model artifacts in HDFS, S3, or any other external storage. Do not use the project filesystem to store large output artifacts.

In general, any project files larger than 50 MB must be part of your project's .gitignore file so that they are not included in *Engines for Experiments and Models* for future experiments/model builds. Note that in case your models require resources that are stored outside the model itself, it is up to you to ensure that these resources are available and immutable as model replicas may be restarted at any time.

Resource Consumption and Scaling

Models should be treated as any other long-running applications that are continuously consuming memory and computing resources. If you are unsure about your resource requirements when you first deploy the model, start with a single replica, monitor its usage, and scale as needed.

If you notice that your models are getting stuck in various stages of the deployment process, check the *Monitoring Active Models* page to make sure that the cluster has sufficient resources to complete the deployment operation.

Security Considerations

As stated previously, models do not impose any limitations on the code they can run. Additionally, models run with the permissions of the user that creates the model (same as sessions and jobs).

Therefore, be conscious of potential data leaks especially when querying underlying data sets to serve predictions.

Cloudera Machine Learning models are not public by default. Each model has an access key associated with it. Only users/applications who have this key can make calls to the model. Be careful with who has permission to view this key.

Cloudera Machine Learning also prints stderr/stdout logs from models to an output pane in the UI. Make sure you are not writing any sensitive information to these logs.

Deployment Considerations

Models deployed using Cloudera Machine Learning Private Cloud are highly available subject to the following limitations:

- Model high availability is dependent on the high availability of the Kubernetes service. If using a third-party Kubernetes service to host CDP Private Cloud, please refer to your chosen provider for precise SLAs.
- In the event that the Kubernetes pod running either the model proxy service or the load balancer becomes unavailable, the Model may be unavailable for multiple seconds during failover.

There can only be one active deployment per model at any given time. This means you should plan for model downtime if you want to deploy a new build of the model or re-deploy with more or fewer replicas.

Keep in mind that models that have been developed and trained using Cloudera Machine Learning are essentially Python or R code that can easily be persisted and exported to external environments using popular serialization formats such as Pickle, PMML, ONNX, and so on.

Related Information

Engines for Experiments and Models

Technical Metrics for Models

Known Issues and Limitations

- Known Issues with Model Builds and Deployed Models
  - Re-deploying or re-building models results in model downtime (usually brief).
  - Re-starting Cloudera Machine Learning does not automatically restart active models. These models must be manually restarted so they can serve requests again.

    Cloudera Bug: DSE-4950
  - Model deployment will fail if your project filesystem is too large for the Git *Engines for Experiments and Models* process. As a general rule, any project files (code, generated model artifacts, dependencies, etc.) larger

than 50 MB must be part of your project's .gitignore file so that they are not included in snapshots for model builds.

- Model builds will fail if your project filesystem includes a .git directory (likely hidden or nested). Typical build stage errors include:

```
Error: 2 UNKNOWN: Unable to schedule build: [Unable to create a checkpoi
nt of current source: [Unable to push sources to git server: ...
```

To work around this, rename the .git directory (for example, NO.git) and re-build the model.

Cloudera Bug: DSE-4657

- JSON requests made to active models should not be more than 5 MB in size. This is because JSON is not suitable for very large requests and has high overhead for binary objects such as images or video. Call the model with a reference to the image or video, such as a URL, instead of the object itself.
- Any external connections, for example, a database connection or a Spark context, must be managed by the model's code. Models that require such connections are responsible for their own setup, teardown, and refresh.
- Model logs and statistics are only preserved so long as the individual replica is active. Cloudera Machine Learning may restart a replica at any time it is deemed necessary (such as bad input to the model).
- (MLLib) The MLLib model.save() function fails with the following sample error. This occurs because the Spark executors on CML all share a mount of /home/cdsw which results in a race condition as multiple executors attempt to write to it at the same time.

```
Caused by:
                  java.io.IOException: Mkdirs failed to create
                  file:/home/cdsw/model.mllib/metadata/_temporary ....
```

Recommended workarounds:

  - Save the model to /tmp, then move it into /home/cdsw on the driver/session.
  - Save the model to either an S3 URL or any other explicit external URL.

- Limitations
  - Scala models are not supported.
  - Spawning worker threads are not supported with models.
  - Models deployed using Cloudera Machine Learning Private Cloud are highly available subject to the following limitations:
    - Model high availability is dependent on the high availability of the Kubernetes service. If using a third-party Kubernetes service to host CDP Private Cloud, please refer to your chosen provider for precise SLAs.
    - In the event that the Kubernetes pod running either the model proxy service or the load balancer becomes unavailable, the Model may be unavailable for multiple seconds during failover.
  - Dynamic scaling and auto-scaling are not currently supported. To change the number of replicas in service, you will have to re-deploy the build.

Related Information

Engines for Experiments and Models

Distributed Computing with Workers

Model Request and Response Formats

Every model function in Cloudera Machine Learning takes a single argument in the form of a JSON-encoded object, and returns another JSON-encoded object as output. This format ensures compatibility with any application accessing the model using the API, and gives you the flexibility to define how JSON data types map to your model's datatypes.

Model Requests

When making calls to a model, keep in mind that JSON is not suitable for very large requests and has high overhead for binary objects such as images or video. Consider calling the model with a reference to the image or video such

as a URL instead of the object itself. Requests to models should not be more than 5 MB in size. Performance may degrade and memory usage increase for larger requests.

Ensure that the JSON request represents all objects in the request or response of a model call. For example, JSON does not natively support dates. In such cases consider passing dates as strings, for example in ISO-8601 format, instead.

For a simple example of how to pass JSON arguments to the model function and make calls to deployed model, see *Creating and Deploying a Model*.

Model Responses

Models return responses in the form of a JSON-encoded object. Model response times depend on how long it takes the model function to perform the computation needed to return a prediction. Model replicas can only process one request at a time. Concurrent requests are queued until a replica is available to process them.

When Cloudera Machine Learning receives a call request for a model, it attempts to find a free replica that can answer the call. If the first arbitrarily selected replica is busy, Cloudera Machine Learning will keep trying to contact a free replica for 30 seconds. If no replica is available, Cloudera Machine Learning will return a model.busy error with HTTP status code 429 (Too Many Requests). If you see such errors, re-deploy the model build with a higher number of replicas.

Model request timeout

You can set the model request timeout duration to a custom value. The default value is 30 seconds. The timeout can be changed if model requests might take more than 30 seconds.

To set the timeout value:

1. As an Admin user, open a CLI.
2. At the prompt, execute the following command. Substitute <value> with the number of seconds to set.

```
kubectl set env deployment model-proxy MODEL_REQUEST_TIMEOUT_SECONDS=<va
lue> -n mlx
```

This edits the `kubeconfig` file and sets a new value for the timeout duration.

Related Information

Creating and Deploying a Model

Workflows for Active Models

Testing Calls to a Model

Cloudera Machine Learning provides two ways to test calls to a model:

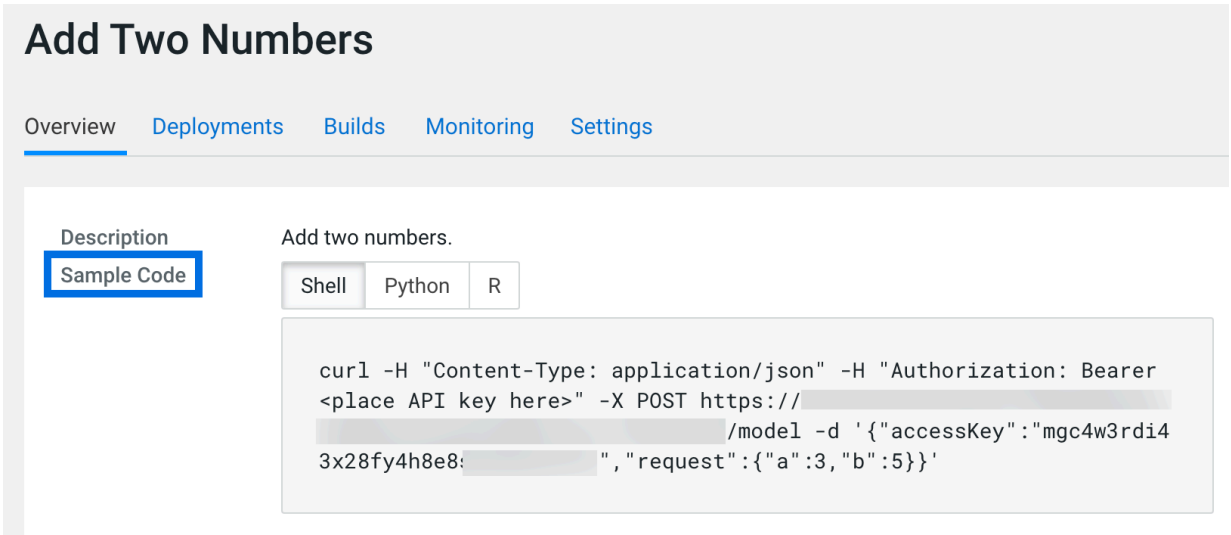
- Test Model Widget

  On each model's Overview page, Cloudera Machine Learning provides a widget that makes a sample call to the deployed model to ensure it is receiving input and returning results as expected.

- Sample Request Strings

  On the model Overview page, Cloudera Machine Learning also provides sample curl and POST request strings that you can use to test calls to the model. Copy/paste the curl request directly into a Terminal to test the call.

  Note that these sample requests already include the example input values you entered while building the model, and the access key required to query the model.



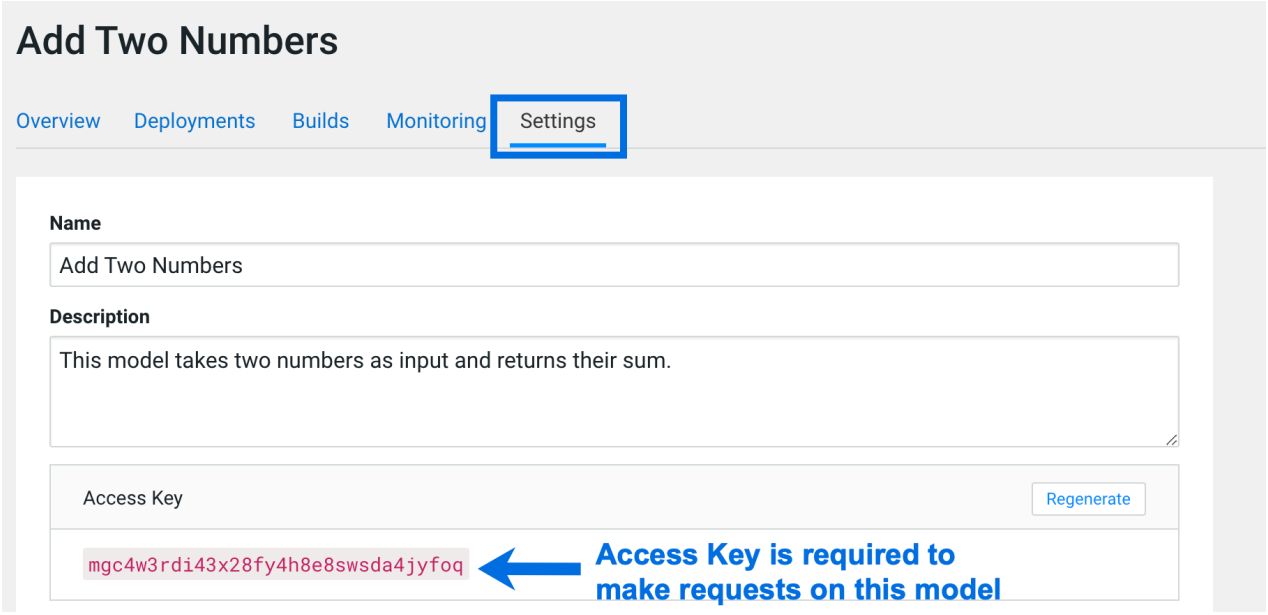
Securing Models

Access Keys for Models

Each model in Cloudera Machine Learning has a unique access key associated with it. This access key is a unique identifier for the model.

Models deployed using Cloudera Machine Learning are not public. In order to call an active model your request must include the model's access key for authentication (as demonstrated in the sample calls above).

To locate the access key for a model, go to the model Overview page and click Settings.



Important:

Only one access key per model is active at any time. If you regenerate the access key, you will need to re-distribute this access key to users/applications using the model.

Alternatively, you can use this mechanism to revoke access to a model by regenerating the access key. Anyone with an older version of the key will not be able to make calls to the model.

API Key for Models

You can prevent unauthorized access to your models by specifying an API key in the “Authorization” header of your model HTTP request. This topic covers how to create, test, and use an API key in Cloudera Machine Learning.

The API key governs the authentication part of the process and the authorization is based on what privileges the users already have in terms of the project that they are a part of. For example, if a user or application has read-only access to a project, then the authorization is based on their current access level to the project, which is “read-only”. If the users have been authenticated to a project, then they can make a request to a model with the API key. This is different from the previously described Access Key, which is only used to identify which model should serve a request.

Enabling authentication

Restricting access using API keys is an optional feature. By default, the “Enable Authentication” option is turned on. However, it is turned off by default for the existing models for backward compatibility. You can enable authentication for all your existing models.

To enable authentication, go to Projects Models Settings and check the Enable Authentication option.

Note: It can take up to five minutes for the system to update.

Generating an API key

If you have enabled authentication, then you need an API key to call a model. If you are not a collaborator on a particular project, then you cannot access the models within that project using the API key that you generate. You need to be added as a collaborator by the admin or the owner of the project to use the API key to access a model.

About this task

There are two types of API keys used in Cloudera Machine Learning:

- API Key: These are used to authenticate requests to a model. You can choose the expiration period and delete them when no longer needed.
- Legacy API Key: This is used in the CDSW-specific internal APIs for CLI automation. This can't be deleted and neither does it expire. This API Key is not required when sending requests to a model.

You can generate more than one API keys to use with your model, depending on the number of clients that you are using to call the models.

Procedure

1. Sign in to Cloudera Machine Learning.
2. Click Settings from the left navigation pane.
3. On the User Settings page, click the API Keys tab.
4. Select an expiry date for the Model API Key, and click Create API keys.

   An API key is generated along with a Key ID.

   If you do not specify an expiry date, then the generated key is active for one year from the current date, or for the duration set by the Administrator. If you specify an expiration date that exceeds the duration value set by the Administrator, you will get an error. The Administrator can set the default duration value at Admin Security Default API keys expiration in days

   Note:

   - The API key is private and ephemeral. Copy the key and the corresponding key ID on to a secure location for future use before refreshing or leaving the page. If you miss storing the key, then you can generate another key.
   - You can delete the API keys that have expired or no longer in use. It can take up to five minutes by the system to take effect.
5. To test the API key:
   a) Navigate to your project and click Models from the left navigation pane.
   b) On the **Overview** page, paste the API key in the API key field that you had generated in the previous step and click Test.

      The test results, along with the HTTP response code and the Replica ID are displayed in the Results table.

      If the test fails and you see the following message, then you must get added as a collaborator on the respective project by the admin or the creator of the project:

      ```
      "User APikey not authorized to access model": "Check APIKEY permissions
      or model authentication permissions"
      ```

Managing API Keys

The admin user can access the list of all the users who are accessing the workspace and can delete the API keys for a user.

About this task

To manage users and their keys:

Procedure

1. Sign in to Cloudera Machine Learning as an admin user.
2. From the left navigation pane, click Admin.
   The **Site Administration** page is displayed.
3. On the **Site Administration** page, click on the Users tab.
   All the users signed under this workspace are displayed.
   The API Keys column displays the number of API keys granted to a user.
4. To delete a API key for a particular user:
   a) Select the user for which you want to delete the API key.
      A page containing the user's information is displayed.
   b) To delete a key, click Delete under the Action column corresponding to the Key ID.
   c) Click Delete all keys to delete all the keys for that user.

   Note: It can take up to five minutes by the system to take effect.

   As a non-admin user, you can delete your own API key by navigating to Settings User Settings API Keys .

Workflows for Active Models

This topic walks you through some nuances between the different workflows available for re-deploying and re-building models.

Active Model - A model that is in the Deploying, Deployed, or Stopping stages.

You can make changes to a model even after it has been deployed and is actively serving requests. Depending on business factors and changing resource requirements, such changes will likely range from changes to the model code itself, to simply modifying the number of CPU/GPUs requested for the model. In addition, you can also stop and restart active models.

Depending on your requirement, you can perform one of the following actions:

Re-deploy an Existing Build

Re-deploying a model involves re-publishing a previously-deployed model in a new serving environment - this is, with an updated number of replicas or memory/CPU/GPU allocation. For example, circumstances that require a re-deployment might include:

- An active model that previously requested a large number of CPUs/GPUs that are not being used efficiently.
- An active model that is dropping requests because it is falling short of replicas.
- An active model needs to be rolled back to one of its previous versions.

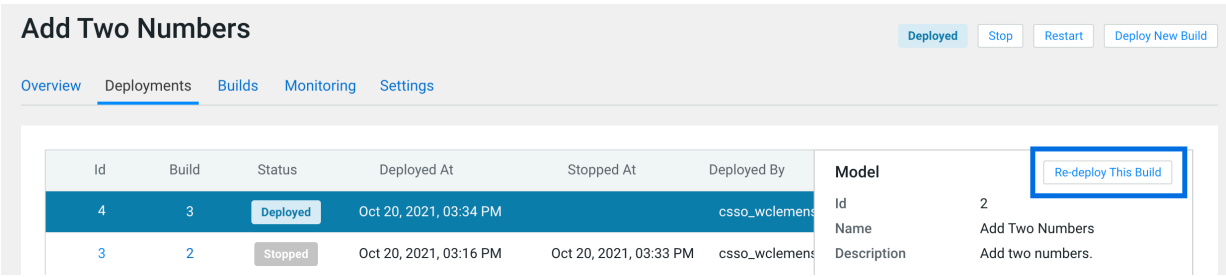
Warning: Currently, Cloudera Machine Learning only allows one active deployment per model. This means when you re-deploy a build, the current active deployment will go offline until the re-deployment process is complete and the new deployment is ready to receive requests. Prepare for model downtime accordingly.

To re-deploy an existing model:

1. Go to the model Overview page.
2. Click Deployments.

3. Select the version you want to deploy and click Re-deploy this Build.

Note:



4. Modify the model serving environment as needed.
5. Click Deploy Model.

Deploy a New Build for a Model

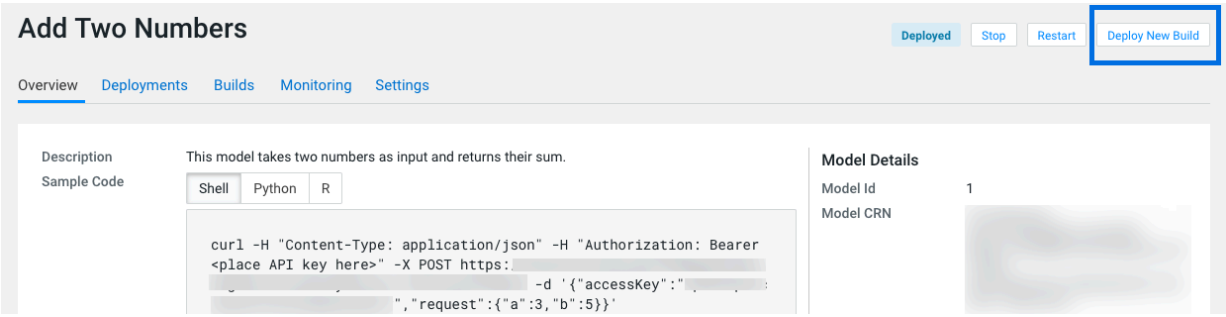
Deploying a new build for a model involves both, re-building the Docker image for the model, and deploying this new build. Note that this is not required if you only need to update the resources allocated to the model. As an example, changes that require a new build might include:

- Code changes to the model implementation.
- Renaming the function that is used to invoke the model.

Warning: Currently, Cloudera Machine Learning does not allow you to create a new build for a model without also deploying it. This combined with the fact that you can only have one active deployment per model means that once the new model is built, the current active deployment will go offline so that the new build can be deployed. Prepare for model downtime accordingly.

To create a new build and deploy it:

1. Go to the model Overview page.
2. Click Deploy New Build.



3. Complete the form and click Deploy Model.

Stop a Model

To stop a model (all replicas), go to the model Overview page and click Stop. Click OK to confirm.

Restart a Model

To restart a model (all replicas), go to the model Overview page and click Restart. Click OK to confirm.

Restarting a model does not let you make any code changes to the model. It should primarily be used as a way to quickly re-initialize or re-connect to resources.

Technical Metrics for Models

You can observe the operation of your models by using charts provided for technical metrics. These charts can help you determine if your models are under- or over-resourced, or are experiencing some problem.

To check the performance of your model, go to Models, click on the model name, and select the Monitoring tab. You can choose to monitor all replicas of the model, or choose a specific replica. You can also select the time and date range to display. Up to two weeks of data is retained.

This tab displays charts for the following technical metrics:

- Requests per Second
- Number of Requests
- Number of Failed Requests
- Model Response Time
- All Model Replica CPU Usage
- All Model Replica Memory Usage
- Model Request & Response Size

All charts share a common time axis (the x axis), so it is easy to correlate cpu and memory usage with model response time or the number of failed requests, for example.

Debugging Issues with Models

This topic describes some common issues to watch out for during different stages of the model build and deployment process.

As a general rule, if your model spends too long in any of the afore-mentioned stages, check the resource consumption statistics for the cluster. When the cluster starts to run out of resources, often models will spend some time in a queue before they can be executed.

Resource consumption by active models on a deployment can be tracked by site administrators on the Admin Models page.

Building

Live progress for this stage can be tracked on the model's Build tab. It shows the details of the build process that creates a new Docker image for the model. Potential issues:

- If you specified a custom build script (cdsw-build.sh), ensure that the commands inside the script complete successfully.
- If you are in an environment with restricted network connectivity, you might need to manually upload dependencies to your project and install them from local files.

Pushing

Once the model has been built, it is copied to an internal Docker registry to make it available to all the Cloudera Machine Learning hosts. Depending on network speeds, your model may spend some time in this stage.

Deploying

If you see issues occurring when Cloudera Machine Learning is attempting to start the model, use the following guidelines to begin troubleshooting:

- Make sure your model code works in a workbench session. To do this, launch a new session, run your model file, and then interactively call your target function with the input object. For a simple example, see the *Creating and Deploying a Model*.

- Ensure that you do not have any syntax errors. For Python, make sure you have the kernel with the appropriate Python version (Python 2 or Python 3) selected for the syntax you have used.
- Make sure that your cdsw-build.sh file provides a complete set of dependencies. Dependencies manually installed during a session on the workbench are not carried over to your model. This is to ensure a clean, isolated, build for each model.
- If your model accesses resources such as data on the CDH cluster or an external database make sure that those resources can accept the load your model may exert on them.

Deployed

Once a model is up and running, you can track some basic logs and statistics on the model's Monitoring page. In case issues arise:

- Check that you are handling bad input from users. If your function throws an exception, Cloudera Machine Learning will restart your model to attempt to get back to a known good state. The user will see an unexpected model shutdown error.

  For most transient issues, model replicas will respond by restarting on their own before they actually crash. This auto-restart behavior should help keep the model online as you attempt to debug runtime issues.
- Make runtime troubleshooting easier by printing errors and output to stderr and stdout. You can catch these on each model's Monitoring tab. Be careful not to log sensitive data here.
- The Monitoring tab also displays the status of each replica and will show if the replica cannot be scheduled due to a lack of cluster resources. It will also display how many requests have been served/dropped by each replica.

Related Information

Engines for Experiments and Models
Creating and Deploying a Model
Technical Metrics for Models

Deleting a Model

Before you begin

Important:

- You must stop all active deployments before you delete a model. If not stopped, active models will continue serving requests and consuming resources even though they do not show up in Cloudera Machine Learning UI.
- Deleted models are not actually removed from disk. That is, this operation will not free up storage space.

Procedure

1. Go to the model Overview Settings .
2. Click Delete Model.

   Deleting a model removes all of the model's builds and its deployment history from Cloudera Machine Learning.

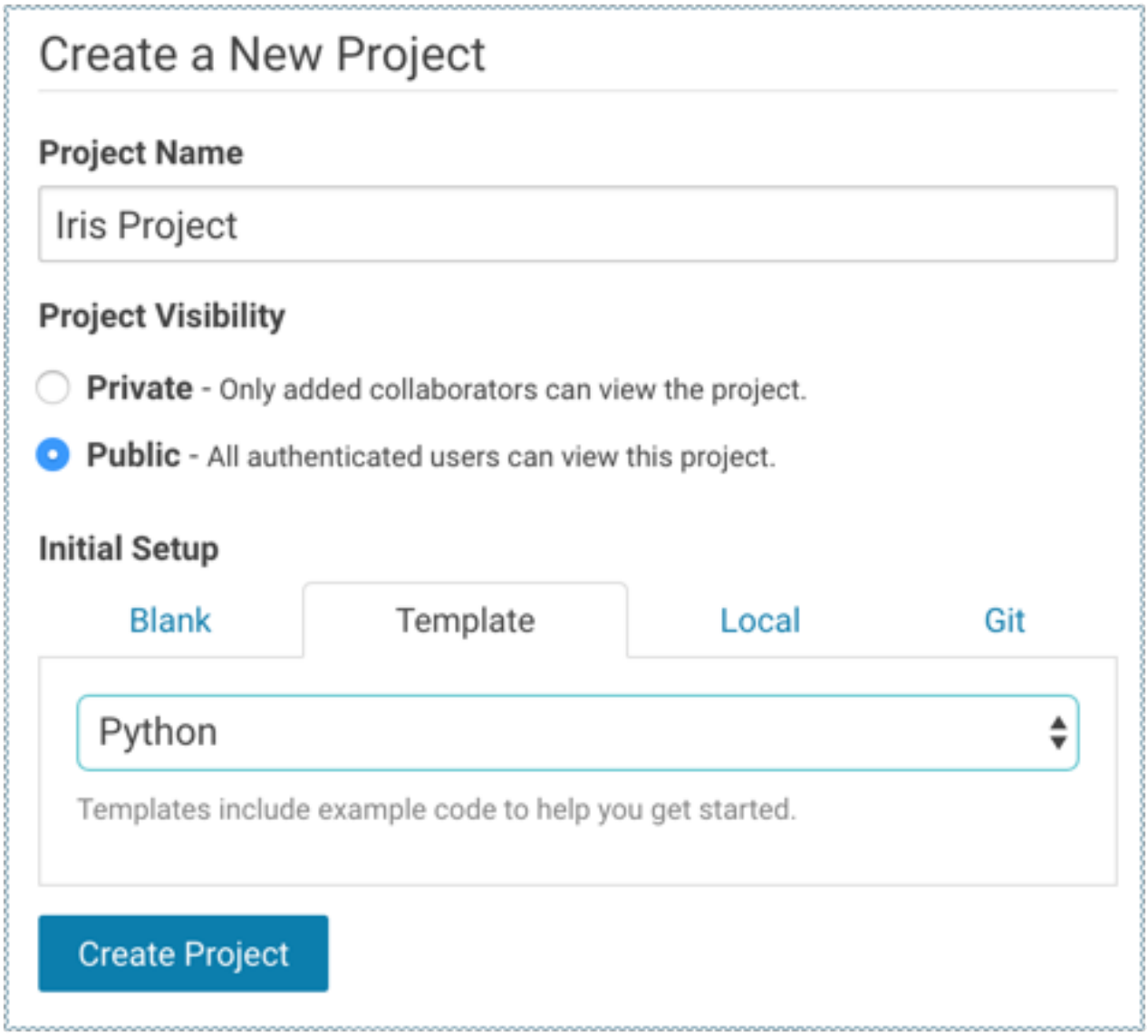
   You can also delete specific builds from a model's history by going to the model's Overview Build page.

Example - Model Training and Deployment (Iris)

This topic uses Cloudera Machine Learning's built-in Python template project to walk you through an end-to-end example where we use experiments to develop and train a model, and then deploy it using Cloudera Machine Learning.

This example uses the canonical Iris dataset from Fisher and Anderson to build a model that predicts the width of a flower's petal based on the petal's length.

The scripts for this example are available in the Python template project that ships with Cloudera Machine Learning. First, create a new project from the Python template:



Once you've created the project, go to the project's Files page. The following files are used for the demo:

- cdsw-build.sh - A custom build script used for models and experiments. Pip installs our dependencies, primarily the scikit-learn library.
- fit.py - A model training example to be run as an experiment. Generates the model.pkl file that contains the fitted parameters of our model.
- predict.py - A sample function to be deployed as a model. Uses model.pkl produced by fit.py to make predictions about petal width.

Related Information

Engines for Experiments and Models

Train the Model

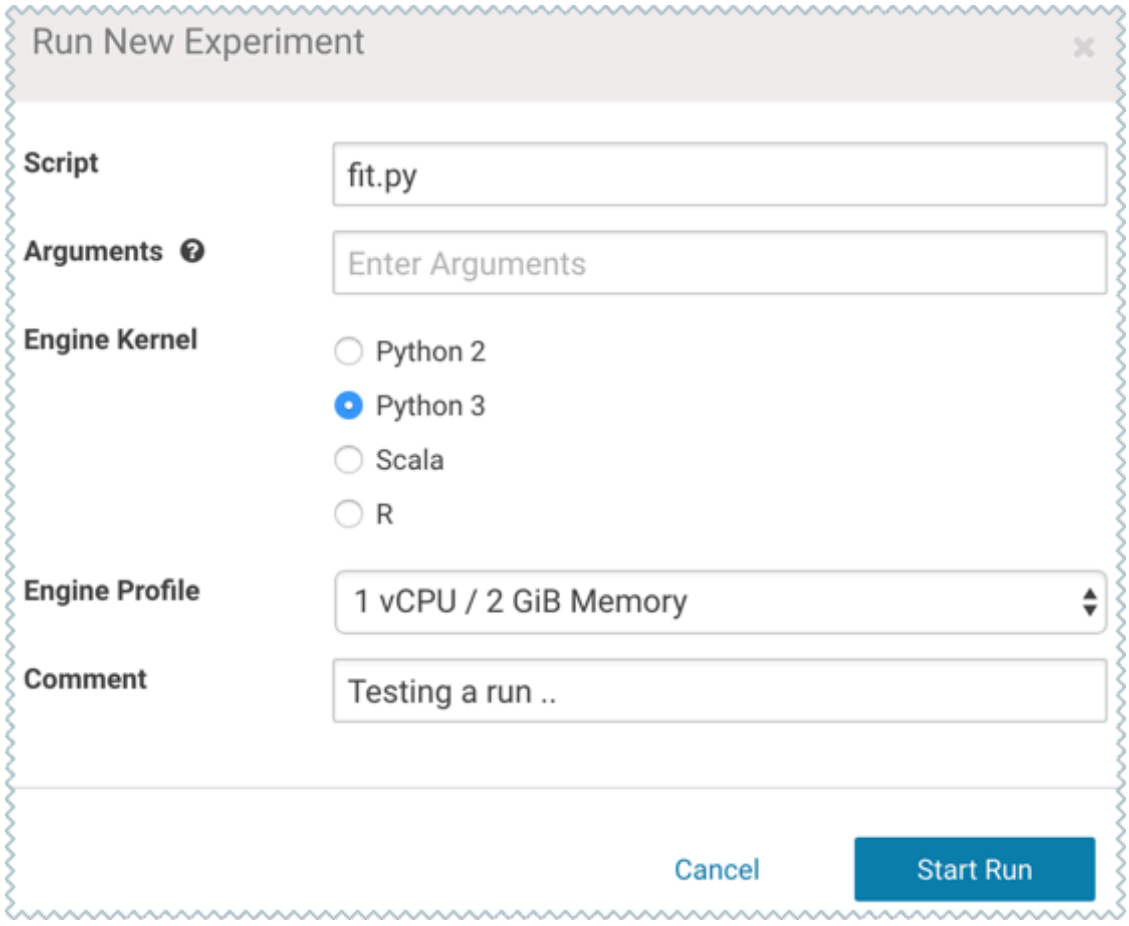
This topic shows you how to run experiments and develop a model using the fit.py file.

About this task

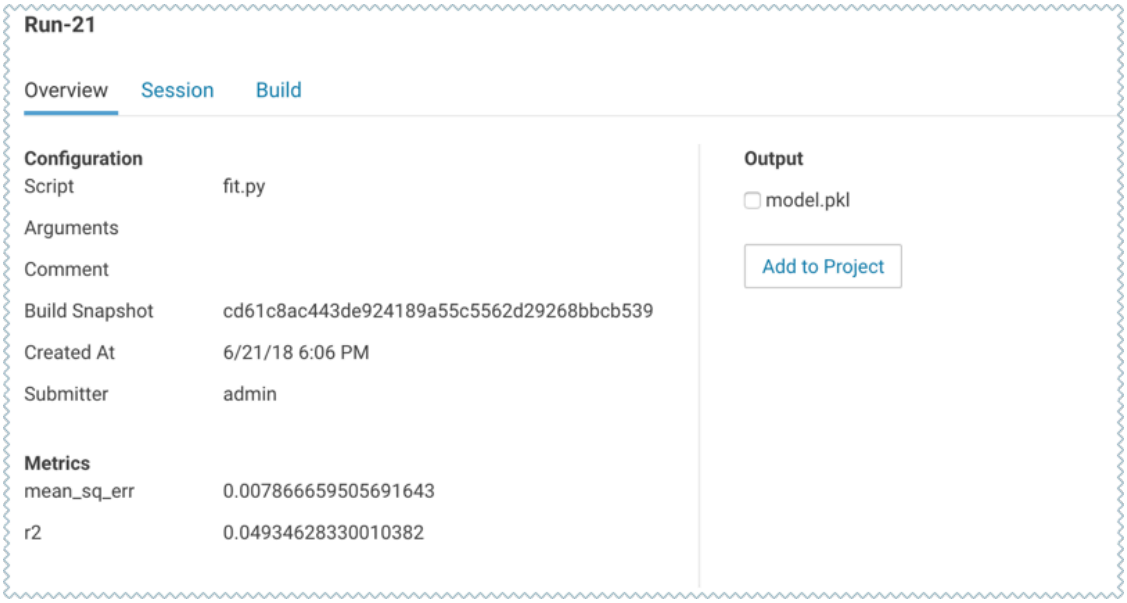
The fit.py script tracks metrics, mean squared error (MSE) and R2, to help compare the results of different experiments. It also writes the fitted model to a model.pkl file.

Procedure

1. Navigate to the Iris project's Overview Experiments page.
2. Click Run Experiment.
3. Fill out the form as follows and click Start Run. Make sure you use the Python 3 kernel.



4. The new experiment should now show up on the Experiments table. Click on the Run ID to go to the experiment's Overview page. The Build and Session tabs display realtime progress as the experiment builds and executes.
5. Once the experiment has completed successfully, go back to its Overview page. The tracked metrics show us that our test set had an MSE of ~0.0078 and an R2 of ~0.0493. For the purpose of this demo, let's consider this an accurate enough model to deploy and use for predictions.



6. Once you have finished training and comparing metrics from different experiments, go to the experiment that generated the best model. From the experiment's Overview page, select the model.pkl file and click Add to Project.
   This saves the model to the project filesystem, available on the project's Files page. We will now deploy this model as a REST API that can serve predictions.

Deploy the Model

This topic shows you how to deploy the model using the predict.py script from the Python template project.

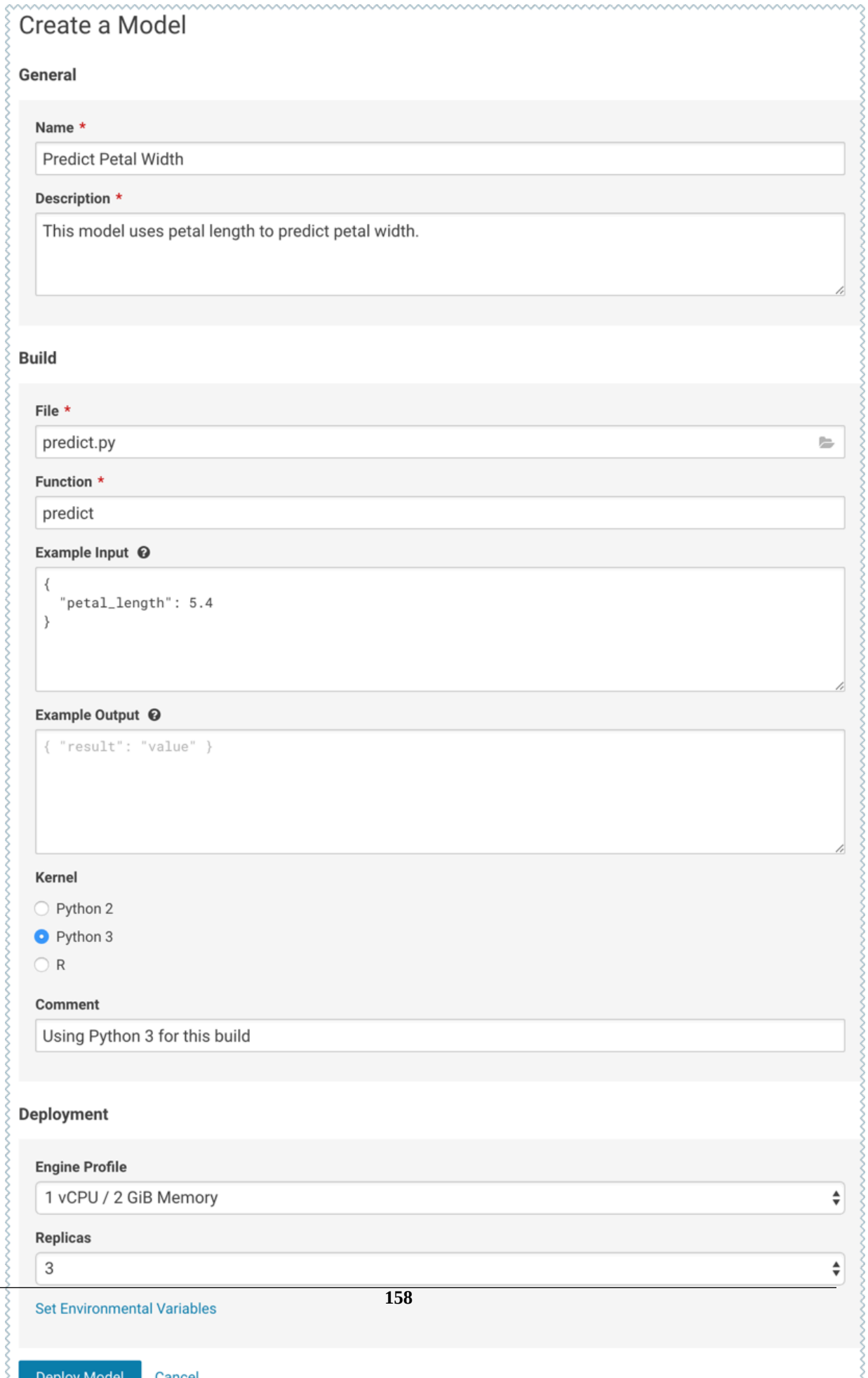
About this task

The predict.py script contains the predict function that accepts petal length as input and uses the model built in the previous step to predict petal width.

Procedure

1. Navigate to the Iris project's Overview Models page.

2. Click New Model and fill out the fields. Make sure you use the Python 3 kernel. For example:


Create a Model
General
Name *
Predict Petal Width
Description *
This model uses petal length to predict petal width.
Build
File *
predict.py
Function *
predict
Example Input
{
"petal_length": 5.4
}
Example Output
{ "result": "value" }
Kernel
Python 2
Python 3
R
Comment
Using Python 3 for this build
Deployment
Engine Profile
1 vCPU / 2 GiB Memory
Replicas
3
Set Environmental Variables

3. Deploy the model.
4. Click on the model to go to its Overview page. As the model builds you can track progress on the Build page. Once deployed, you can see the replicas deployed on the Monitoring page.
5. To test the model, use the Test Model widget on the model's Overview page.

Enabling model governance

You must enable governance to capture and view information about your ML projects, models, and builds centrally from Apache Atlas (Data Catalog) for a given environment. If you do not select this option while provisioning workspaces, then integration with Atlas won't work.

About this task

Procedure

1. Go to Cloudera Machine Learning and click Provision Workspace on the top-right corner.
2. Enter the workspace name and other details.
3. Click Advanced Options.
4. Select Enable Governance.

ML Governance Requirements

You must ensure that the following requirements are satisfied in order to enable ML Governance on Private Cloud.

The following services on CDP must be enabled:

- Kafka
- Ranger
- Solr
- Atlas

On Cloudera Manager (CM), ensure that the following are enabled in the base cluster for Cloudera Manager:

- Auto-TLS
- Kerberos (either MIT or AD)

Registering training data lineage using a linking file

The Machine Learning (ML) projects, model builds, model deployments, and associated metadata are tracked in Apache Atlas, which is available in the environment's SDX cluster. You can also specify additional metadata to be tracked for a given model build. For example, you can specify metadata that links training data to a project through a special file called the linking file (lineage.yaml).

The lineage.yaml file describes additional metadata and the lineage relationships between the project's models and training data. You can use a single lineage.yaml file for all the models within the project.

Note: Your lineage file should be present in your project before you create a model build. The lineage file is parsed and metadata is attached during the model build process.

1. Create a YAML file in your ML project called lineage.yaml.

   If you have used a template to create your project, a lineage.yaml file should already exist in your project.
2. Insert statements in the file that describe the relationships you want to track between a model and the training data. You can include additional descriptive metadata through key-value pairs in a metadata section.

YAML	YAML Structure	Description
Model name	Top-level entry	A ML model name associated with the current project. There can be more than one model per linking file.
hive_table_qualified_names	Second-level entry	This pre-defined key introduces sequence items that list the names of Hive tables used as training data.
Table names	Sequence items	The qualified names of Hive tables used as training data enclosed in double quotation marks. Qualified names are of the format *db-name.table-name@cluster-name*
metadata	Second-level entry	This pre-defined key introduces additional metadata to be included in the Atlas representation of the relationship between the model and the training data.
key:value	Third-level entries	Key-value pairs that describe information about how this data is used in the model. For example, consider including the query text that is used to extract training data or the name of the training file used.

The following example linking file shows entries for two models in your project: modelName1 and modelName2:

```
modelName1:                                 # the name of your model
  hive_table_qualified_names:               # this is a predefined key to link to
                                            # training data
    - "db.table1@namespace"                 # the qualifiedName of the hive_table
                                            # object representing training data
    - "db.table2@ns"
  metadata:                                 # this is a predefined key for
                                            # additional metadata

    key1: value1
    key2: value2
    query: "select id, name from table"     # suggested use case: query used to
                                            # extract training data
    training_file: "fit.py"                 # suggested use case: training file
                                            # used
modelName2:                                 # multiple models can be specified in
                                            # one file
  hive_table_qualified_names:
    - "db.table2@ns"
```

Viewing lineage for a model deployment in Atlas

You can view the lineage information for a particular model deployment and trace it back to the specific data that was used to train the model through the Atlas' Management Console.

Procedure

1. Navigate to Management Console Environments , select your environment, and then under Quick Links select Atlas.
2. Search for ml_model_deployment. Click the model deployment of your interest.
3. Click the Lineage tab to see a visualization of lineage information for the particular model deployment and trace it back to the specific data that was used to train the model.

   You can also search for a specific table, click through to its Lineage tab and see if the table has been used in any model deployments.

Enabling model metrics

Metrics are used to track the performance of the models. When you enable model metrics while creating a workspace, the metrics are stored in a scalable metrics store. You can track individual model predictions and analyze metrics using custom code.

About this task

Procedure

1. Go to Cloudera Machine Learning and click Provision Workspace on the top-right corner.
2. Enter the workspace name and other details.
3. Click Advanced Options.
4. Select Enable Model Metrics.

   If you want to connect to an external (custom) Postgres database, then specify the details in the additional optional arguments that are displayed. If you do not specify these details, a managed Postgres database will be used to store the metrics.

Tracking model metrics without deploying a model

Cloudera recommends that you develop and test model metrics in a workbench session before actually deploying the model. This workflow avoids the need to rebuild and redeploy a model to test every change.

Metrics tracked in this way are stored in a local, in-memory datastore instead of the metrics database, and are forgotten when the session exits. You can access these metrics in the same session using the regular metrics API in the cdsw.py file.

The following example demonstrates how to track metrics locally within a session, and use the read_metrics function to read the metrics in the same session by querying by the time window.

To try this feature in the local development mode, use the following files from the Python template project:

- use_model_metrics.py
- predict_with_metrics.py

The predict function from the predict_with_metrics.py file shown in the following example is similar to the function with the same name in the predict.py file. It takes input and returns output, and can be deployed as a model.
But unlike the function in the predict.py file, the predict function from the predict_with_metrics.py file tracks mathematical metrics. These metrics can include information such as input, output, feature values, convergence

metrics, and error estimates. In this simple example, only input and output are tracked. The function is equipped to track metrics by applying the decorator cdsw.model_metrics.

```
@cdsw.model_metrics
def predict(args):
  # Track the input.
  cdsw.track_metric("input", args)

  # If this model involved features, ie transformations of the
  # raw input, they could be tracked as well.
  # cdsw.track_metric("feature_vars", {"a":1,"b":23})

  petal_length = float(args.get('petal_length'))
  result = model.predict([[petal_length]])
  # Track the output.
  cdsw.track_metric("predict_result", result[0][0])
  return result[0][0]
```

You can directly call this function in a workbench session, as shown in the following example:

```
predict(
{"petal_length": 3}
)
```

You can fetch the metrics from the local, in-memory datastore by using the regular metrics API. To fetch the metrics, set the dev keyword argument to True in the use_model_metrics.py file. You can query the metrics by model, model build, or model deployment using the variables cdsw.dev_model_crn and cdsw.dev_model_build_crn or cdsw.dev _model_deploy_crn respectively.

For example:

```
end_timestamp_ms=int(round(time.time() * 1000))
cdsw.read_metrics(model_deployment_crn=cdsw.dev_model_deployment_crn,
start_timestamp_ms=0,
end_timestamp_ms=end_timestamp_ms,
dev=True)
```

where CRN denotes Cloudera Resource Name, which is a unique identifier from CDP, analogous to Amazon's ARN.

Tracking metrics for deployed models

When you have finished developing your metrics tracking code and the code that consumes the metrics, simply deploy the predict function from predict_with_metrics.py as a model. No code changes are necessary.

Calls to read_metrics, track_delayed_metrics, and track_aggregate_metrics need to be changed to take the CRN of the deployed model, build or deployment. These CRNs can be found in the model's **Overview** page.

Calls to the call_model function also requires the model's access key (model_access_key in use_model_metrics.py) from the model's **Settings** page. If authentication has been enabled for the model (the default), a model API key for the user (model_api_token in use_model_metrics.py) is also required. This can be obtained from the user's **Settings** page.

Analytical Applications

This topic describes how to use an ML workspace to create long-running web applications.

About this task:

This feature gives data scientists a way to create ML web applications/dashboards and easily share them with other business stakeholders. Applications can range from single visualizations embedded in reports, to rich dashboard solutions such as Tableau. They can be interactive or non-interactive.

Applications stand alongside other existing forms of workloads in CML (sessions, jobs, experiments, models). Like all other workloads, applications must be created within the scope of a project. Each application is launched within its own isolated engine. Additionally, like models, engines launched for applications do not time out automatically. They will run as long as the web application needs to be accessible by any users and must be stopped manually when needed.

Before you begin:

Testing applications before you deploy

Before you deploy an application using the steps described here, make sure your application has been thoroughly tested. You can use sessions to develop, test, and debug your applications. You can test web apps by embedding them in sessions as described here: https://docs.cloudera.com/machine-learning/1.4.1/projects/topics/ml-embedded-web-apps.html.

For CML UI

1. Go to a project's Overview page.
2. Click Applications.
3. Click New Application.
4. Fill out the following fields.
   - Name: Enter a name for the application.
   - Subdomain: Enter a subdomain that will be used to construct the URL for the web application. For example, if you use test-app as the subdomain, the application will be accessible at test-app.<ml-workspace-domain-name>.

     Subdomains should be valid DNS hostname characters: letters from a to z, digits from 0 to 9, and the hyphen.
   - Description: Enter a description for the application.
   - Script: Select a script that hosts a web application on either CDSW_READONLY_PORT or CDSW_APP_PORT. Applications running on either of these ports are available to any users with at least read access to the project. The Python template project includes an entry.py script that you can use to test this out.

     Note: CML does not prevent you from running an application that allows a read-only user (i.e. Viewers) to modify files belonging to the project. It is up to you to make the application truly read-only in terms of files, models, and other resources belonging to the project.
   - Engine Kernel and Resource Profile: Select the kernel and computing resources needed for this application.
   - Set Environment Variables: Click Set Environment Variables, enter the name and value for the new application variable, and click Add.

     If there is a conflict between the project-level and application-level environment variables, the application-level environment variables override the project-level environment variables.
5. Click Create Application.

For CML APIv2

To create an application using the API, refer to this example:

Here is an example of using the Application API.

```
application_request = cmlapi.CreateApplicationRequest(
     name = "application_name",
     description = "application_description",
     project_id = project_id,
     subdomain = "application-subdomain",
     kernel = "python3",
```

```
      script = "entry.py",
      environment = {"KEY": "VAL"}
)
app = client.create_application(
      project_id = project_id,
      body = application_request
)
```

Results:

In a few minutes, you should see the application status change to Running on the Applications page. Click on the name of the application to access the web application interface.

What to do next:

You can Stop, Restart, or Delete an application from the Applications page.

If you want to make changes to an existing application, click Overview under the application name. Then go to the Settings tab to make any changes and update the application.

Securing Applications

You can provide access to Applications via either the CDSW_APP_PORT or the CDSW_READONLY_PORT. Any user with read or higher permissions to the project is able to access an application served through either port.

- Securing project resources

  CML applications are accessible by any user with read-only or higher permissions to the project. The creator of the application is responsible for managing the level of permissions the application users have on the project through the application. CML does not actively prevent you from running an application that allows a read-only user (i.e. Viewers) to modify files belonging to the project.
- Public Applications

  By default, authentication for applications is enforced on all ports and users cannot create public applications. If desired, the Admin user can allow users to create public applications that can be accessed by unauthenticated users.

  To allow users to create public applications on an ML workspace:

  1. As an Admin user, turn on the feature flag in Admin Security by selecting Allow applications to be configured with unauthenticated access.
  2. When creating a new application, select Enable Unauthenticated Access.
  3. For an existing application, in Settings select Enable Unauthenticated Access.

  To prevent all users from creating public applications, go to Admin Security and deselect Allow applications to be configured with unauthenticated access. After one minute, all existing public applications stop being publicly accessible.
- Transparent Authentication

  CML can pass user authentication to an Application, if the Application expects an authorized request. The REMO TE-USER field is used for this task.

Limitations with Analytical Applications

This topic lists all the limitations associated with the Applications feature.

- Port availability

  Cloudera Machine Learning exposes only 2 ports per workload. Therefore, you can run a maximum of 2 web applications simultaneously, on these ports:

  - CDSW_APP_PORT
  - CDSW_READONLY_PORT

  By default, third-party browser-based editors run on CDSW_APP_PORT. Therefore, for projects that are already using browser-based editors, you are left with only one other port to run applications on: CDSW_READONLY_PORT.

Monitoring applications

You can monitor the CPU and memory usage of deployed applications.

To view the Monitoring Applications UI, in Applications, select the Monitoring Applications icon.

The Monitoring Applications UI provides the following information to help you monitor application health. The information is shown both as an instantaneous value and as a time series.

- CPU Usage: Shows the cpu usage as a percentage of total available CPU resources, and as amount of vCPU.
- Memory Usage: Shows the memory usage as a percentage of total available memory, and as amount of memory.

You can also select to view a specific application or all applications, and also select the time period to view.

Creating a Job

This topic describes how to automate analytics workloads with a built-in job and pipeline scheduling system that supports real-time monitoring, job history, and email alerts.

A job automates the action of launching an engine, running a script, and tracking the results, all in one batch process. Jobs are created within the purview of a single project and can be configured to run on a recurring schedule. You can customize the engine environment for a job, set up email alerts for successful or failed job runs, and email the output of the job to yourself or a colleague.

Jobs are created within the scope of a project. When you create a job, you will be asked to select a script to run as part of the job, and create a schedule for when the job should run. Optionally, you can configure a job to be dependent on another existing job, thus creating a pipeline of tasks to be accomplished in a sequence. Note that the script files and any other job dependencies must exist within the scope of the same project.

For CML UI

1. Navigate to the project for which you want to create a job.
2. On the left-hand sidebar, click Jobs.
3. Click New Job.
4. Enter a Name for the job.
5. In Script, select a script to run for this job by clicking on the folder icon. You will be able to select a script from a list of files that are already part of the project. To upload more files to the project, see *Managing Project Files*.
6. In Arguments, enter command-line arguments to provide to the script.

   This feature only works with R or Python engines.
7. Depending on the code you are running, select an Engine Kernel for the job from one of the following option: Python 3.

8. Select a Schedule for the job runs from one of the following options.
   - Manual - Select this option if you plan to run the job manually each time.
   - Recurring - Select this option if you want the job to run in a recurring pattern every X minutes, or on an hourly, daily, weekly or monthly schedule. Set the recurrence interval with the drop-down buttons.

     As an alternative, select Use a cron expression to enter a Unix-style cron expression to set the interval. The expression must have five fields, specifying the minutes, hours, day of month, month, and day of week. If the cron expression is deselected, the schedule indicated in the drop-down settings takes effect.
   - Dependent - Use this option when you are building a pipeline of jobs to run in a predefined sequence. From a dropdown list of existing jobs in this project, select the job that this one should depend on. Once you have configured a dependency, this job will run only after the preceding job in the pipeline has completed a successful run.
9. Select an Resource Profile to specify the number of cores and memory available for each session.
10. Enter an optional timeout value in minutes.
11. Click Set environment variables if you want to set any values to override the overall project environment variables.
12. Specify a list of Job Report Recipients to whom you can send email notifications with detailed job reports for job success, failure, or timeout. You can send these reports to yourself, your team (if the project was created under a team account), or any other external email addresses.
13. Add any Attachments such as the console log to the job reports that will be emailed.
14. Click Create Job.

    You can use the API v2 to schedule jobs from third partly workflow tools. For details, see *Using the Jobs API* as well as the *CML APIv2* tab.

For CML APIv2

To create a job using the API, follow the code below:

```
job_body = cmlapi.CreateJobRequest()

# name and script
job_body.name = "my job name"
job_body.script = "pi.py"

# arguments
job_body.arguments = "arg1 arg2 \"all arg 3\""

# engine kernel
job_body.kernel = "python3" # or "r", or "scala"

# schedule
# manual by default
# for recurring/cron:
job_body.schedule = "* * * * 5" # or some valid cron string

# for dependent (don't set both parent_job_id and schedule)
job_body.parent_job_id = "abcd-1234-abcd-1234"

# resource profile (cpu and memory can be floating point for partial)
job_body.cpu = 1 # one cpu vcore
job_body.memory = 1 # one GB memory
job_body.nvidia_gpu = 1 # one nvidia gpu, cannot do partial gpus

# timeout
job_body.timeout = 300 # this is in seconds

# environment
job_body.environment = {"MY_ENV_KEY": "MY_ENV_VAL", "MY_SECOND_ENV_KEY":
"MY_SECOND_ENV_VAL"}
```

```
# attachment
job_body.attachments = ["report/1.txt", "report/2.txt"] # will attach /
home/cdsw/report/1.txt and /home/cdsw/report/2.txt to emails

# After setting the parameters above, create the job:
client = cmlapi.default_client("host", "api key")
client.create_job(job_body, project_id="id of project to create job in")
```

For some more examples of commands related to jobs, see: *Using the Jobs API*.

Related Information

Managing Project Files
Using the Jobs API
Legacy Jobs API (Deprecated)

Creating a Pipeline

This topic describes how to create a scheduled pipeline of jobs within a project.

About this task

As data science projects mature beyond ad hoc scripts, you might want to break them up into multiple steps. For example, a project may include one or more data acquisition, data cleansing, and finally, data analytics steps. For such projects, Cloudera Machine Learning allows you to schedule multiple jobs to run one after another in what is called a pipeline, where each job is dependent on the output of the one preceding it.

The Jobs overview presents a list of all existing jobs created for a project along with a dependency graph to display any pipelines you've created. Job dependencies do not need to be configured at the time of job creation. Pipelines can be created after the fact by modifying the jobs to establish dependencies between them. From the job overview, you can modify the settings of a job, access the history of all job runs, and view the session output for individual job runs.

Let's take an example of a project that has two jobs, Read Weblogs and Write Weblogs. Given that you must read the data before you can run analyses and write to it, the Write Weblogs job should only be triggered after the Read Weblogs job completes a successful run. To create such a two-step pipeline:

Procedure

1. Navigate to the project where the Read Weblogs and Write Weblogs jobs were created.
2. Click Jobs.
3. From the list of jobs, select Write Weblogs.
4. Click the Settings tab.
5. Click on the Schedule dropdown and select Dependent. Select Read Weblogs from the dropdown list of existing jobs in the project.
6. Click Update Job.

Viewing Job History

This topics shows you how to view the history for jobs run within a project.

Procedure

1. Navigate to the project where the job was created.
2. Click Jobs.
3. Select the relevant job.

4. Click the History tab. You will see a list of all the job runs with some basic information such as who created the job, run duration, and status. Click individual runs to see the session output for each run.

Legacy Jobs API (Deprecated)

This topic demonstrates how to use the legacy API to launch jobs.

Cloudera Machine Learning exposes a legacy REST API that allows you to schedule jobs from third-party workflow tools. You must authenticate yourself before you can use the legacy API to submit a job run request. The Jobs API supports HTTP Basic Authentication, accepting the same users and credentials as Cloudera Machine Learning.

Note: The Jobs API is now deprecated. See *CML API v2* and API v2 usage for the successor API.

Legacy API Key Authentication

Cloudera recommends using your legacy API key for requests instead of your actual username/password so as to avoid storing and sending your credentials in plaintext. The legacy API key is a randomly generated token that is unique to each user. It must be treated as highly sensitive information because it can be used to start jobs via the API. To look up your Cloudera Machine Learning legacy API key:

1. Sign in to Cloudera Machine Learning.
2. From the upper right drop-down menu, switch context to your personal account.
3. Click Settings.
4. Select the API Key tab.

The following example demonstrates how to construct an HTTP request using the standard basic authentication technique. Most tools and libraries, such as Curl and Python Requests, support basic authentication and can set the required Authorization header for you. For example, with curl you can pass the legacy API key to the --user flag and leave the password field blank.

```
curl -v -XPOST http://cdsw.example.com/api/v1/<path_to_job> --user
 "<LEGACY_API_KEY>:"
```

To access the API using a library that does not provide Basic Authentication convenience methods, set the request's Authorization header to Basic <*LEGACY_API_KEY_encoded_in_base64*>. For example, if your API key is uysg xtj7jzkps96njextnxxmq05usp0b, set Authorization to Basic dXlzZ3h0ajdqemtwczk2bmpleHRueHhtcTA1dXNwMG I6.

Starting a Job Run Using the API

Once a job has been created and configured through the Cloudera Machine Learning web application, you can start a run of the job through the legacy API. This will constitute sending a POST request to a job start URL of the form: http://cdsw.example.com/api/v1/projects/<$USERNAME>/<$PROJECT_NAME>/jobs/<$JOB_ID>/start.

To construct a request, use the following steps to derive the username, project name, and job ID from the job's URL in the web application.

1. Log in to the Cloudera Machine Learning web application.
2. Switch context to the team/personal account where the parent project lives.
3. Select the project from the list.
4. From the project's Overview, select the job you want to run. This will take you to the job Overview page. The URL for this page is of the form: http://cdsw.example.com/<$USERNAME>/<$PROJECT_NAME>/jobs/<$JOB_ID>.

5. Use the $USERNAME, $PROJECT_NAME, and $JOB_ID parameters from the job Overview URL to create the following job start URL: http://cdsw.example.com/api/v1/projects/<$USERNAME>/<$PROJECT_NAME>/jobs/<$JOB_ID>/start.

   For example, if your job Overview page has the URL http://cdsw.example.com/alice/sample-project/jobs/123, then a sample POST request would be of the form:

   ```
   curl -v -XPOST http://cdsw.example.com/api/v1/projects/alice/sample-proj
   ect/jobs/123/start    \
   --user "<API_KEY>:" --header "Content-type: application/json"
   ```

   Note that the request must have the Content-Type header set to application/json, even if the request body is empty.

Setting Environment Variables

You can set environment variables for a job run by passing parameters in the API request body in a JSON-encoded object with the following format.

```
{
    "environment": {
        "ENV_VARIABLE": "value 1",
        "ANOTHER_ENV_VARIABLE": "value 2"
    }
}
```

The values set here will override the defaults set for the project and the job in the web application. This request body is optional and can be left blank.

Be aware of potential conflicts with existing defaults for environment variables that are crucial to your job, such as PATH and the CML variables.

Sample Job Run

As an example, let's assume user Alice has created a project titled Risk Analysis. Under the Risk Analysis project, Alice has created a job with the ID, 208. Using curl, Alice can use her API Key (uysgxtj7jzkps96njextnxxmq05usp0b) to create an API request as follows:

```
curl -v -XPOST http://cdsw.example.com/api/v1/projects/alice/risk-analysis/j
obs/208/start      \
--user "uysgxtj7jzkps96njextnxxmq05usp0b:" --header "Content-type: applicat
ion/json"          \
--data '{"environment": {"START_DATE": "2017-01-01", "END_DATE": "2017-01-
31"}}'
```

In this example, START_DATE and END_DATE are environment variables that are passed as parameters to the API request in a JSON object.

In the resulting HTTP request, curl automatically encodes the Authorization request header in base64 format.

```
* Connected to cdsw.example.com (10.0.0.3) port 80 (#0)
* Server auth using Basic with user 'uysgxtj7jzkps96njextnxxmq05usp0b'
> POST /api/v1/projects/alice/risk-analysis/jobs/21/start HTTP/1.1
> Host: cdsw.example.com
> Authorization: Basic dXlzZ3h0ajdqemtwczk2bmpleHRueHhtcTA1dXNwMGI6
> User-Agent: curl/7.51.0
> Accept: */*
> Content-type: application/json
>
< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Content-Type: application/json; charset=utf-8
```

```
< Date: Mon, 10 Jul 2017 12:00:00 GMT
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
<
{
  "engine_id": "cwg6wclmg0x482u0"
}
```

You can confirm that the job was started by going to the Cloudera Machine Learning web application.

Starting a Job Run Using Python

To start a job run using Python, Cloudera recommends using Requests, an HTTP library for Python; it comes with a convenient API that makes it easy to submit job run requests to Cloudera Machine Learning. Extending the Risk Analysis example from the previous section, the following sample Python code creates an HTTP request to run the job with the job ID, 208.

Python 2

```
# example.py

import requests
import json

HOST = "http://cdsw.example.com"
USERNAME = "alice"
API_KEY = "uysgxtj7jzkps96njextnxxmq05usp0b"
PROJECT_NAME = "risk-analysis"
JOB_ID = "208"

url = "/".join([HOST, "api/v1/projects", USERNAME, PROJECT_NAME, "jobs",
JOB_ID, "start"])
job_params = {"START_DATE": "2017-01-01", "END_DATE": "2017-01-31"}
res = requests.post(
    url,
    headers = {"Content-Type": "application/json"},
    auth = (API_KEY,""),
    data = json.dumps({"environment": job_params})
)

print "URL", url
print "HTTP status code", res.status_code
print "Engine ID", res.json().get('engine_id')
```

When you run the code, you should see output of the form:

```
python example.py
```

```
URL http://cdsw.example.com/api/v1/projects/alice/risk-analysis/jobs/208/sta
rt
HTTP status code 200
Engine ID r11w5q3q589ryg9o
```

Limitations

- Cloudera Machine Learning does not support changing your legacy API key, or having multiple API keys.
- Currently, you cannot create a job, stop a job, or get the status of a job using the Jobs API.

Related Information

API v2 usage

Basic Access Authentication
Creating a Pipeline
Environment Variables
CML API v2

Distributed Computing with Workers

Cloudera Machine Learning provides basic support for launching multiple engine instances, known as workers, from a single interactive session. Any R or Python session can be used to spawn workers. These workers can be configured to run a script (e.g. a Python file) or a command when they start up.

Workers can be launched using the launch_workers function. Other supported functions are list_workers and stop _workers. Output from all the workers is displayed in the workbench console of the session that launched them. These workers are terminated when the session exits.

Using Workers for Machine Learning

The simplest example of using this feature would involve launching multiple workers from a session, where each one prints 'hello world' and then terminates right after. To extend this example, you can remove the print command and configure the workers to run a more elaborate script instead. For example, you can set up a queue of parameters (inputs to a function) in your main interactive session, and then configure the workers to run a script that pulls parameters off the queue, applies a function, and keeps doing this until the parameter queue is empty. This generic idea can be applied to multiple real-world use-cases. For example, if the queue is a list of URLs and the workers apply a function that scrapes a URL and saves it to a database, CML can easily be used to do parallelized web crawling.

Hyperparameter optimization is a common task in machine learning, and workers can use the same parameter queue pattern described above to perform this task. In this case, the parameter queue would be a list of possible values of the hyperparameters of a machine learning model. Each worker would apply a function that trains a machine learning model. The workers run until the queue is empty, and save snapshots of the model and its performance.

Workers API

This section lists the functions available as part of the workers API.

Launch Workers

Launches worker engines into the cluster.

Syntax

```
launch_workers(n, cpu, memory, nvidia_gpu=0, kernel="python3", s
cript="", code="", env={})
```

Parameters

- n (int) - The number of engines to launch.
- cpu (float) - The number of CPU cores to allocate to the engine.
- memory (float) - The number of gigabytes of memory to allocate to the engine.
- nvidia_gpu (int, optional) - The number of GPU's to allocate to the engine.
- kernel (str, optional) - The kernel. Can be "r", "python2", "python3" or "scala". This parameter is only available for projects that use legacy engines.
- script (str, optional) - The name of a Python source file the worker should run as soon as it starts up.
- code (str, optional) - Python code the engine should run as soon as it starts up. If a script is specified, code will be ignored.
- env (dict, optional) - Environment variables to set in the engine.

Example Usage

Python

```
import cdsw
workers = cdsw.launch_workers(n=2, cpu=0.2, memory=0.5, code="pr
int('Hello from a CDSW Worker')")
```

R

```
library("cdsw")
workers <- launch.workers(n=2, cpu=0.2, memory=0.5, env="", cod
e="print('Hello from a CML Worker')")
```

Note: The env parameter has been defined due to a bug that appears when parsing the launch.workers function. When not defined, the env parameter is serialized internally into a format that is incompatible with Cloudera Machine Learning. This bug does not affect the Python engine.

List Workers

Returns all information on all the workers in the cluster.

Syntax

```
list_workers()
```

Stop Workers

Stops worker engines.

Syntax

```
stop_workers(*worker_id)
```

Parameter

- worker_id (int, optional) - The ID numbers of the worker engines that must be stopped. If an ID is not provided, all the worker engines on the cluster will be stopped.

Worker Network Communication

This section demonstrates some trivial examples of how two worker engines communicate with the master engine.

Workers are a low-level feature to help use higher level libraries that can operate across multiple hosts. As such, you will generally want to use workers only to launch the backends for these libraries.

To help you get your workers or distributed computing framework components talking to one another, every worker engine run includes an environmental variable CML_MASTER_IP with the fully addressable IP of the master engine. Every engine has a dedicated IP access with no possibility of port conflicts.

For instance, the following are trivial examples of two worker engines talking to the master engine.

R

From the master engine, the following master.r script will launch two workers and accept incoming connections from them.

```
# master.r

library("cdsw")
# Launch two CML workers. These are engines that will run in
# the same project, run a given code or script, and exit.
workers <- launch.workers(n=2, cpu=0.2, memory=0.5, env="", script="worker.
r")
# Accept two connections, one from each worker. Workers will
# run worker.r.
```

```
for(i in c(1,2)) {
  # Receive a message from each worker and return a response.
  con <- socketConnection(host="0.0.0.0", port = 6000, blocking=TRUE, ser
ver=TRUE, open="r+")
  data <- readLines(con, 1)
  print(paste("Server received:", data))
  writeLines("Hello from master!", con)
  close(con)
}
```

The workers will run the following worker.r script and respond to the master.

```
# worker.r

print(Sys.getenv("CML_MASTER_IP"))
con <- socketConnection(host=Sys.getenv("CML_MASTER_IP"), port = 6000, bl
ocking=TRUE, server=FALSE, open="r+")
write_resp <- writeLines("Hello from Worker", con)
server_resp <- readLines(con, 1)
print(paste("Worker received:  ", server_resp))
close(con)
```

Python

From the master engine, the following master.py script will launch two workers and accept incoming connections from them.

```
# master.py
import cdsw, socket

# Launch two CDSW workers. These are engines that will run in
# the same project, run a given code or script, and exit.
workers = cdsw.launch_workers(n=2, cpu=0.2, memory=0.5, script="worker.py")

# Listen on TCP port 6000
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("0.0.0.0", 6000))
s.listen(1)

# Accept two connections, one from each worker. Workers will
# run worker.py.
conn, addr = s.accept()
for i in range(2):
    # Receive a message from each worker and return a response.
    data = conn.recv(20)
    if not data: break
    print("Master received:", data)
    conn.send("Hello From Server!".encode())
conn.close()
```

The workers will run the following worker.py script and respond to the master.

```
# worker.py
import os, socket

# Open a TCP connection to the master.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((os.environ["CDSW_MASTER_IP"], 6000))

# Send some data and receive a response.
s.send("Hello From Worker!".encode())
data = s.recv(1024)
```

```
s.close()

print("Worker received:", data)
```

Applied ML Prototypes (AMPs)

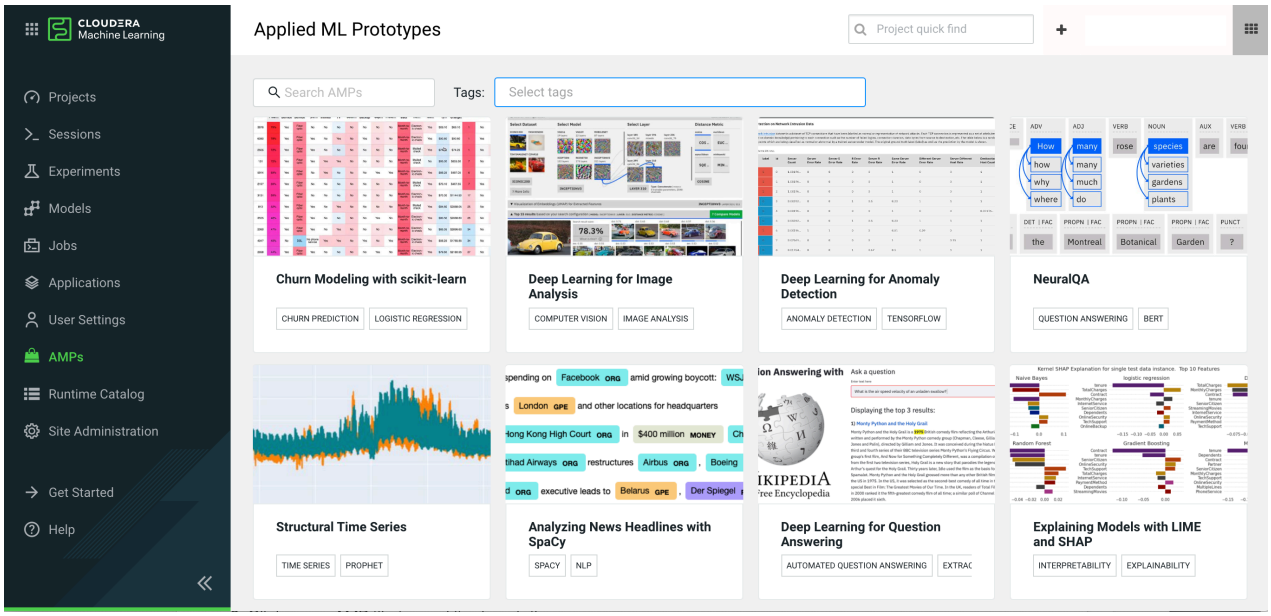
Applied ML Prototypes (AMPs) provide reference example machine learning projects in Cloudera Machine Learning. More than simplified quickstarts or tutorials, AMPs are fully-developed expert solutions created by Cloudera's research arm, Fast Forward Labs.

Note: AMPs are not supported for air gap environments.

These solutions to common problems in the machine learning field demonstrate how to fully use the power of Cloudera Machine Learning. AMPs show you how to create Cloudera Machine Learning projects to solve your own use cases.

AMPs are available to install and run from the Cloudera Machine Learning user interface. As new AMPs are developed, they will become available to you for your study and use.

Note: In an airgapped installation, the default AMPs catalog included at installation and default AMPs may be inaccessible. Consider creating an custom AMPs catalog. See *Custom AMP Catalog* for more information.



Using AMPs

It's simple to get started with AMPs.

1. Log in to your Cloudera Machine Learning workspace, and in the left panel click AMPs.
2. Click on an AMP tile to read its description.
3. Click Configure Project and provide any configuration values required by the AMP. The Description field explains how to determine these configuration values. After you click Launch Project, the installation process may take several minutes.
4. When the installation is complete, click Overview to read the documentation for the AMP and explore the code and project structure.

Note: If nothing appears in the AMPs panel, an administrator may need to reconfigure and refresh the catalog. In Site Administration AMPs , click Refresh. The administrator can also refresh periodically to add newly developed AMPs to the panel.

Related Information

Custom AMP Catalog

Creating New AMPs

One great use for AMPs is to showcase reference examples specific to your business by creating your own AMPs in-house. Once a data science project has been built in Cloudera Machine Learning, you can package it and have the Cloudera Machine Learning Admin add it to the AMP Catalog.

Each individual AMP requires a project metadata file, which defines the environmental resources needed by the AMP, and the setup steps to install the AMP in a Cloudera Machine Learning workspace. See AMP Project Specification for details.

Note: You can store your AMPs in a git repo hosted on Github, Github Enterprise, or GitLab servers (not limited to github.com or gitlab.com.)

Additionally, only simple authentication is supported, such as passing an API key, or including the username and password, as part of the URL. If additional authentication steps are required, then that git host is not supported.

You can also look at an example for a Cloudera AMP, such as: .project-metadata.yaml.

Related Information

AMP Project Specification

Custom AMP Catalog

An AMP catalog is a collection of AMPs that can be added to a workspace as a group. Cloudera Machine Learning ships with the Cloudera AMP catalog, containing AMPs developed by Cloudera Fast Forward Labs, but you can also create and add your own catalog, containing AMPs developed by your organization.

To create an AMP catalog, you need to create a YAML file called the catalog file. This file is hosted on GitHub or another git server. This could be either a public or a private git server.

The catalog file contains information about each AMP in the catalog, and provides a link to the AMP repository itself. The catalog file specification is shown in Catalog File Specification.

You can also look at the Cloudera catalog file for an example. To view the file, click directly on the URL for Cloudera in Catalog Sources.

For more details on creating the AMPs that you will include in your catalog, see Creating New AMPs.

One use case you might consider is creating a fork of the Cloudera AMP catalog, in order to host it internally. In this case, you will need to edit the URLs in the catalog and project metadata files to point to the appropriate internal resources.

Related Information

Catalog File Specification

Creating New AMPs

Add a catalog

The collection of AMPs available to end users can draw from one or more sources. For example, you might have an internal company catalog in addition to the default Cloudera catalog. The catalog must be provided with a catalog file and one or more project metadata YAML files.

About this task

Specify Catalog File URL if your git hosting service allows you to access the raw content of the repo without authenticating. (That is, the source files can be retrieved with a curl command, and do not require logging into a web page). Otherwise, specify the Git Repository URL. To use a git repository as a catalog source, the catalog file and the AMP files must be in a repository that can be cloned with `git clone` without authentication.

Procedure

1. As an Administrator, go to Site Administration AMPs .
2. Select Git Repository URL or Catalog File URL to specify a new source. Paste or enter the URL to the new source, and file name for the catalog file if necessary.
3. Click Add Source.

   The catalog YAML file is loaded, and the projects found there are displayed in Catalog Entries.
4. If there are projects that are not yet ready for use, or that should not be displayed in the catalog, deselect Enabled in the Catalog Entries.

Catalog File Specification

The Catalog file is a YAML file that contains descriptive information and metadata for the displaying the AMP projects in the Project Catalog.

Fields

Fields are in snake_case. Each project in the catalog uses the following fields:

Field Name	Type	Example	Description
name	string	name: Cloudera	Required. Name of the catalog, displayed as Source in the Prototype Catalog tab.
entries	string	entries:	Required. Contains the entries for each project.
title	string	title: Churn Modeling	Required. The title of the AMP, as displayed in the Prototype Catalog.
label	string	label: churn-prediction	Required.
short_description	string	short_description: Build an scikit-learn model...	Required. A short description of the project. Appears on the project tile in the Prototype Catalog.
long_description	string	long_description: >- This project demonstrates...	Required. A longer description that appears when the user clicks on the project tile.
image_path	string	image_path: >- https://raw .git...	Required. Path to the image file that displays in the Prototype Catalog.
tags	string	tags: - Churn Prediction - Logistic Regression	Required. For sorting in the Prototype Catalog pane.

Field Name	Type	Example	Description
git_url	string	git_url: "https:..."	Required. Path to the git repository for the project.
is_prototype	boolean	is_prototype: true	Optional. Indicates the AMP should be displayed in the Prototype Catalog. Use if coming_soon is not used.
coming_soon	boolean	coming_soon: true	Optional. Displays the AMP in the Prototype Catalog with a “COMING SOON” watermark. Use if is_prototype is not used.

Example:

```
name: Cloudera

entries:
   - title: Churn Modeling with scikit-learn
     label: churn-prediction
     short_description: Build an scikit-learn model to predict churn using
 customer telco data.
     long_description: >-
     This project demonstrates how to build a logistic regression classific
ation model to predict the probability
     that a group of customers will churn from a fictitious telecommunicatio
ns company. In addition, the model is
     interpreted using a technique called Local Interpretable Model-agnos
tic Explanations (LIME). Both the logistic
     regression and LIME models are deployed using CML's real-time model dep
loyment capability and interact with a
     basic Flask-based web application.
     image_path: >-
      https://raw.githubusercontent.com/cloudera/Applied-ML-Prototypes/maste
r/images/churn-prediction.jpg
    tags:
      - Churn Prediction
      - Logistic Regression
      - Explainability
      - Lime
    git_url: "https://github.com/cloudera/CML_AMP_Churn_Prediction"
    is_prototype: true
```

AMP Project Specification

AMP projects include a project metadata file that provides configuration and setup details. These details may include environment variables and tasks to be run on startup.

YAML File Specification # Version 1.0

The project metadata file is a YAML file. It must be placed in your project's root directory, and must be named .project-metadata.yaml. The specifications for this file are listed below. You can also look at an example for one of the Cloudera AMPs, such as:.project-metadata.yaml.

Fields

Fields for this YAML file are in snake_case. String fields are generally constrained by a fixed character size, for example string(64) is constrained to contain at most 64 characters. Click Show to see the list of fields.

Field Name	Type	Example	Description
name	string(200)	ML Demo	Required: The name of this project prototype. Prototype names do not need to be unique.
description	string(2048)	This demo shows off some cool applications of ML.	Required: A description for this project prototype.
author	string(64)	Cloudera Engineer	Required: The author of this prototype (can be the name of an individual, team, or organization).
date	date string	"2020-08-11"	The date this project prototype was last modified. It should be in the format: "YYYY-MM-DD" (quotation marks are required).
specification_version	string(16)	0.1	Required: The version of the YAML file specification to use.
prototype_version	string(16)	1.0	Required: The version of this project prototype.
shared_memory_limit	number	0.0625	Additional shared memory in GB available to sessions running in this project. The default is 0.0625 GB (64MB).
environment_variables	environment variables object	See below	Global environment variables for this project prototype.
feature_dependencies	feature_dependencies	See below	A list of feature dependencies of this AMP. A missing dependency in workspace blocks the creation of the AMP.
engine_images	engine_images	See below	Engine images to be used with the AMP. What's specified here is a recommendation and it does not prevent the user from launching an AMP

Field Name	Type	Example	Description
			with non recommended engine images.
runtimes	runtimes	See below	Runtimes to be used with the AMP. What's specified here is a recommendation and it does not prevent the user from launching an AMP with non recommended runtimes.
tasks	task list	See below	A sequence of tasks, such as running Jobs or deploying Models, to be run after project import.

Example

```
name: ML Demo
description: >-
This demo shows off some cool applications of ML.
author: Cloudera Engineer
date: '2020-08-11T17:40:00.839Z'
specification_version: 1.0
environment_variables:
...
tasks:
...
```

Environment variables object

The YAML file can optionally define any number of global environment variables for the project under the environment field. This field is an object, containing keys representing the names of the environment variables, and values representing details about those environment variables. Click Show to see the list of fields in the Environment variables object.

Field Name	Type	Example	Description
default	string	"3"	The default value for this environment variable. Users may override this value when importing this project prototype.
description	string	The number of Model replicas, 3 is standard for redundancy.	A short description explaining this environment variable.
required	boolean	true	Whether the environment variable is required to have a non-empty value, the default is false.

Example: This example creates four environment variables.

```
environment_variables:
  AWS_ACCESS_KEY:
    default: ""
    description: "Access Key ID for accessing S3 bucket"
  AWS_SECRET_KEY:
    default: ""
    description: "Secret Access Key for accessing S3 bucket"
    required: true
  HADOOP_DATA_SOURCE:
    default: ""
    description: "S3 URL to large data set"
    required: false
  MODEL_REPLICAS:
    default: "3"
    description: "Number of model replicas, 3 is standard for redundancy"
    required: true
```

Feature Dependencies

AMPs might depend on some optional features of a workspace. The feature_dependencies field accepts a list of such features. Unsatisified feature dependencies prevent the AMP from being launched in a workspace, and display an appropriate error message. The supported feature dependencies are as follows:

- model_metrics

Runtimes Specification

The runtimes field accepts a list of runtimes objects defined as follows. This Runtimes specification can be added per task or per project.

```
- editor: the_name_of_the_editor # case-sensitive string required. e.g. Work
bench, Jupyter, etc. (how it appears in the UI)
  kernel: the_kernel # case-sensitive string required. e.g. Python 3.6, Pyth
on 3.8, R 3.6, etc. (how it appears in the UI)
  edition: the_edition # case-sensitive string required. e.g. Standard, N
vidia GPU, etc. (how it appears in the UI)
  version: the_short_version # case-sensitive string optional. e.g. 2021.
03, 2021.05, etc. (how it appears in the UI)
  addons: the_list_addons_needed # list of case-sensitive strings optional.
 e.g Spark 2.4.7 - CDP 7.2.11 - CDE 1.13, etc. (how it appears in the UI)
```

This example specifies the Runtimes the Workbench version for Python 3.8.

```
runtimes:
  - editor: Workbench
    kernel: Python 3.8
    edition: Standard
    addons: ['Spark 2.4.7 - CDP 7.2.11 - CDE 1.13']
```

Engine Images Specification

The engine_images field accepts a list of engine_image objects defined as follows:

```
- image_name: the_name_of_the_engine_image # string (required)
  tags: # list of strings (optional)
    - the_tag_of_engine_image
    - ...
```

This example specifies the official engine image with version 11 or 12:

```
engine_images:
  - image_name: engine
    tags:
      - 12
      - 11
```

This example specifies the most recent version of the dataviz engine image in the workspace:

```
engine_images:
  - image_name: cmldataviz
  - image_name: cdswdataviz
```

Note that when specifying CDV images, both cmldataviz and cdswdataviz must be specified. When tags are not specified, the most recent version of the engine image with the matching name is recommended. The following rule is used to determine the most recent engine_image with the matching name:

Official Engine (engine) and CDV (cmldataviz and cdswdataviz) images

Since the officially released engine images follow semantic versioning (where a newer version is always larger than any older version, when compared with >), the most recent engine image is the one with the largest tag. For example, engine:14 will be recommended over engine:13 and cmldataviz:6.3.4-b13 is recommended over cmldataviz:6.2.1-b12.

Custom engine images

There is no way for Cloudera Machine Learning to determine the rules for customer custom engine image tags, and therefore there is no reliable way to determine the most recent custom engine image. You should use the engine image that has the correct matching name and has the newest id. The newest id means that the engine image is the most recently added engine image.

Task list

This defines a list of tasks that can be automatically run on project import. Each task will be run sequentially in the order they are specified in this YAML file. Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	create_job	Required: The type of task to be executed. See below for a list of allowed types.
short_summary	string	Creating a Job that will do a task.	A short summary of what this task is doing.
long_summary	string	Creating a Job that will do this specific task. This is important because it leads up to this next task.	A long summary of what this task is doing.

Jobs

Create Job

Example

```
- type: create_job
  name: howdy
```

```
  entity_label: howdy
  script: greeting.py
  arguments: Ofek 21
  short_summary: Creating a job that will greet you.
  environment_variables:
  SAMPLE_ENVIRONMENT_VARIABLE: CREATE/RUN_JOB
  kernel: python3
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	create_job	Required: Must be create_job.
name	string	howdy	Required: Job name.
entity_label	string	howdy	Required: Uniquely identifies this job for future tasks, i.e. run_job tasks. Entity labels must be lowercase alphanumeric, and may contain hyphens or underscores.
script	string	greeting.py	Required: Script for this Job to run.
kernel	string	python3	Required: What kernel this Job should use. Acceptable values are python2, python3, r, and scala. Note that scala might not be supported for every cluster.
arguments	string	Ofek 21	Command line arguments to be given to this Job when running.
environment_variables	environment variables object	See above	See above
cpu	number	1.0	The amount of CPU virtual cores to allocate for this Job, the default is 1.0.
memory	number	1.0	The amount of memory in GB to allocate for this Job, the default is 1.0.
gpu	integer	0	The amount of GPU to allocate for this Job, the default is 0.

Field Name	Type	Example	Description
timeout	integer	10	The amount of time in minutes to wait before timing out this Job, the default is 10.
timeout_kil	boolean	false	Whether or not to stop this Job when it times out, the default is false.

Run Job

Example run job task:

```
- type: run_job
  entity_label: howdy
  short_summary: Running the job that will greet you.
  long_summary: >-
    Running the job that will greet you. It will greet you by the name
    which is the first and only command line argument.
```

Most Job run tasks should just contain the type and entity_label fields. Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	run_job	Required: Must be run_job.
entity_label	string	howdy	Required: Must match an entity_label of a previous create_job task.

However, they can optionally override previously defined fields. Click Show to see the list of fields.

Field Name	Type	Example	Description
script	string	greeting.py	Required: Script for this Job to run.
kernel	string	python3	Required: What kernel this Job should use. Acceptable values are python2, pyth on3, r, and scala. Note that scala might not be supported for every cluster.
arguments	string	Ofek 21	Command line arguments to be given to this Job when running.
environment_variables	environment variables object	See above	See above

Field Name	Type	Example	Description
cpu	number	1.0	The amount of CPU virtual cores to allocate for this Job, the default is 1.0
memory	number	1.0	The amount of memory in GB to allocate for this Job, the default is 1.0.
gpu	integer	0	The amount of GPU to allocate for this Job, the default is 0.
shared_memory_limit	number	0.0625	Limits the additional shared memory in GB that can be used by this Job, the default is 0.0625 GB (64MB).

Models

Note: All models have authentication disabled, so their access key alone is enough to interact with them.

Resources object

Models may define a resources object which overrides the amount of resources to allocate per Model deployment.

Click Show to see the list of fields.

Field Name	Type	Example	Description
cpu	number	1.0	The number of CPU virtual cores to allocate per Model deployment.
memory	number	2.0	The amount of memory in GB to allocate per Model deployment.
gpu	integer	0	The amount of GPU to allocate per Model deployment.

For example:

```
resources:
  cpu: 1
  memory: 2
```

Replication policy object

Models may define a replication policy object which overrides the default replication policy for Model deployments.

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	fixed	Must be fixed if present.
num_replicas	integer	1	The number of replicas to create per Model deployment.

For example:

```
replication_policy:
  type: fixed
  num_replicas: 1
```

Model examples list

Models may include examples, which is a list of objects containing a request and response field, each containing a valid object inside, as shown in the example:

```
examples:
  - request:
      name: Ofek
      age: 21
    response:
      greeting: Hello Ofek (21)
  - request:
      name: Jimothy
      age: 43
    response:
      greeting: Hello Coy (43)
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
request	string	See above	Required: An example request object.
num_replicas	string	See above	Required: The response to the above example request object.

Create Model

Example:

```
- type: create_model
  name: Say hello to me
  entity_label: says-hello
  description: This model says hello to you
  short_summary: Deploying a sample model that you can use to greet you
  access_key_environment_variable: SHTM_ACCESS_KEY
  default_resources:
  cpu: 1
  memory: 2
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	create_model	Required: Must be crea te_model.
name	string	Say hello to me	Required: Model name
entity_label	string	says-hello	Required: Uniquely identifies this model for future tasks, i.e. build_mo del and deploy_model tasks. Entity labels must be lowercase alphanumeric, and may contain hyphens or underscores.
access_key_environment_variable	string	SHTM_ACCESS_KEY	Saves the model's access key to an environment variable with the specified name.
default_resources	resources object	See above	The default amount of resources to allocate per Model deployment.
default_replication_policy	replication policy object	See above	The default replication policy for Model deployments.
description	string	This model says hello to you	Model description.
visibility	string	private	The default visibility for this Model.

Build Model

Example

```
- type: build_model
  entity_label: says-hello
  comment: Some comment about the model
  examples:
    - request:
        name: Ofek
        age: 21
    response:
      greeting: Hello Ofek (21)
  target_file_path: greeting.py
  target_function_name: greet_me
  kernel: python3
  environment_variables:
    SAMPLE_ENVIRONMENT_VARIABLE: CREATE/BUILD/DEPLOY_MODEL
```

Field Name	Type	Example	Description
type	string	build_model	Required: Must be buil d_model.
entity_label	string	says-hello	Required: Must match an entity_label of a previous create_model task.
target_file_path	string	greeting.py	Required: Path to file that will be run by Model.
target_function_name	string	greet_me	Required: Name of function to be called by Model.
kernel	string	python3	What kernel this Model should use. Acceptable values are python2, pyth on3, r, and scala. Note that scala might not be supported for every cluster.
comment	string	Some comment about the model	A comment about the Model.
examples	model examples list	See above	A list of request/response example objects.
environment_variables	environment variables object	See above	See above

Deploy Model

Example:

```
- type: deploy_model
  entity_label: says-hello
  environment_variables:
  SAMPLE_ENVIRONMENT_VARIABLE: CREATE/BUILD/DEPLOY_MODEL
```

Most deploy model tasks should just contain the type and entity_label fields. Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	deploy_model	Required: Must be depl oy_model.
entity_label	string	says-hello	Required: Must match an entity_label of a previous deploy_model task.

However, they can optionally override previously defined fields. Click Show to see the list of fields.

Field Name	Type	Example	Description
cpu	number	1.0	The number of CPU virutal cores to allocate for this Model deployment.
memory	number	2.0	The amount of memory in GB to allocate for this Model deployment.
gpu	integer	0	The amount of GPU to allocate for this Model deployment.
replication_policy	replication policy object	See above	The replication policy for this Model deployment.
environment_variables	environment variables object	See above	Overrides environment variables for this Model deployment.

Applications

Start Application

Example:

```
- type: start_application
  subdomain: greet
  script: greeting.py
  environment_variables:
    SAMPLE_ENVIRONMENT_VARIABLE: START_APPLICATION
  kernel: python3
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	start_application	Required: Must be start_application.
subdomain	string	greet	Required: Application subdomain, which must be unique per Application, and must be alphanumeric and hyphen-delimited. Application subdomains are also converted to lowercase.
kernel	string	python3	Required: What kernel this Application should use. Acceptable values are python2, python3, r, and scala. Note that scala might

Field Name	Type	Example	Description
			not be supported for every cluster.
entity_label	string	greeter	Uniquely identifies this application for future tasks. Entity labels must be lowercase alphanumeric, and may contain hyphens or underscores.
script	string	greeting.py	Script for this Application to run.
name	string	Greeter	Application name, defaults to 'Untitled application'.
description	string	Some description about the Application	Application description, defaults to 'No description for the app'.
cpu	number	1.0	The number of CPU virutal cores to allocate for this Application.
memory	number	1.0	The amount of memory in GB to allocate for this Application.
gpu	integer	0	The amount of GPU to allocate for this Application.
shared_memory_limit	number	0.0625	Limits the additional shared memory in GB that can be used by this application, the default is 0.0625 GB (64MB).
environment_variables	environment variables object	See above	See above

Experiments

Run Experiment

Example:

```
- type: run_experiment
  script: greeting.py
  arguments: Ofek 21
  kernel: python3
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	run_experiment	Required: Must be run_experiment.
script	string	greeting.py	Required: Script for this Experiment to run.
entity_label	string	test-greeter	Uniquely identifies this experiment for future tasks. Entity labels must be lowercase alphanumeric, and may contain hyphens or underscores.
arguments	string	Ofek 21	Command line arguments to be given to this Experiment when running.
kernel	string	python3	What kernel this Experiment should use. Acceptable values are python2, python3, r, and scala. Note that scala might not be supported for every cluster.
comment	string	Comment about the experiment	A comment about the Experiment.
cpu	number	1.0	The amount of CPU virtual cores to allocate for this Experiment.
memory	number	1.0	The amount of memory in GB to allocate for this Experiment.
gpu	number	0	The amount of GPU to allocate for this Experiment.

Sessions

Run Sessions

Example:

```
- type: run_session
  name: How to be greeted interactively
  code: |
    import os
    os.environ['SAMPLE_ENVIRONMENT_VARIABLE'] = 'SESSION'

    !python3 greeting.py Ofek 21

    import greeting
```

```
    greeting.greet_me({'name': 'Ofek', 'age': 21})
  kernel: python3
  memory: 1
  cpu: 1
  gpu: 0
```

Click Show to see the list of fields.

Field Name	Type	Example	Description
type	string	run_session	Required: Must be run_session.
	string	See above for code, greeting.py for script	Required: Either the code or script field is required to exist for the run Session task, not both. code is a direct block of code that will be run by the Session, while script is a script file that will be executed by the Session.
kernel	string	python3	Required: What kernel this Session should use. Acceptable values are python2, python3, r, and scala. Note that scala might not be supported for every cluster.
cpu	number	1.0	Required: The amount of CPU virtual cores to allocate for this Session.
memory	number	1.0	Required: The amount of memory in GB to allocate for this Session.
entity_label	string	greeter	Uniquely identifies this session for future tasks. Entity labels must be lowercase alphanumeric, and may contain hyphens or underscores.
name	string	How to be greeted interactively	Session name.
gpu	integer	0	The amount of GPU to allocate for this Session.

Host names required by AMPs

If you are using a non-transparent proxy in AWS, then you need to allow the following host names in order for AMPs to work.

- *.storage.googleapis.com
- *.raw.githubusercontent.com
- *.pypi.org
- *.pythonhosted.org
- *.github.com

These host names should be specified in the proxy, along with any other endpoints that are needed for your workspace.

Managing Users

This topic describes how to manage an ML workspace as a site administrator. Site administrators can monitor and manage all user activity across a workspace, add new custom engines, and configure certain security settings.

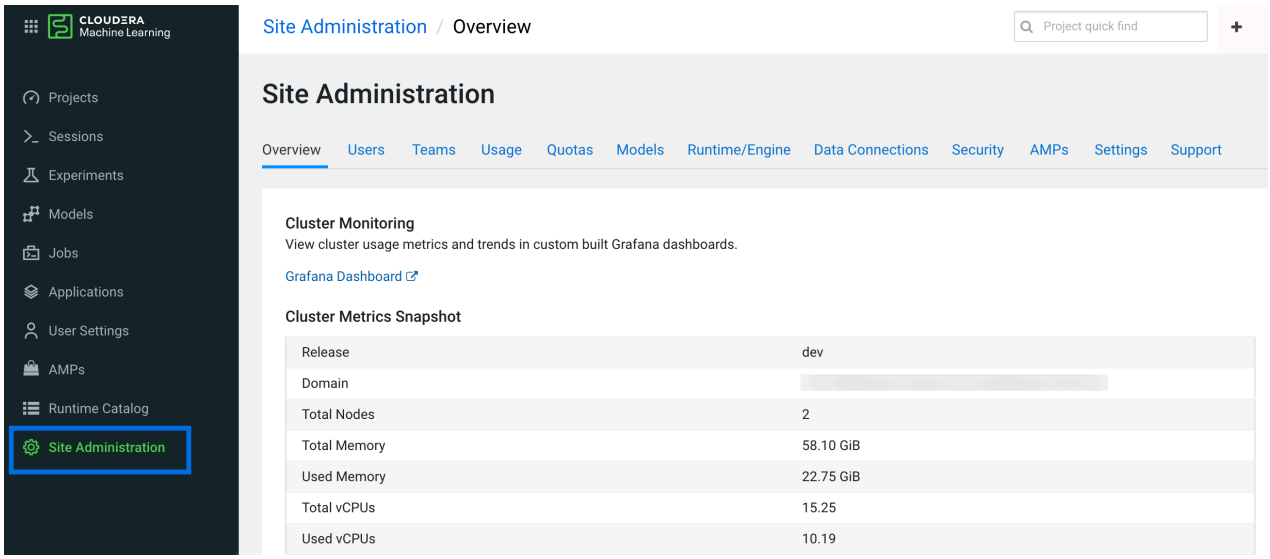
By default, the first user that logs in to a workspace must always be a site administrator. That is, they should have been granted the MLAdmin role by a CDP PowerUser.

Note: On Private Cloud, the corresponding role is EnvironmentAdmin.

Important: Site administrators have complete access to all activity on the deployment. This includes access to all teams and projects on the deployment, even if they have not been explicitly added as team members or collaborators.

Only site administrators have access to a Site Administration dashboard that can be used to manage the workspace. To access the site administrator dashboard:

1. Go to the Cloudera Machine Learning web application and log in as a site administrator.
2. On the left sidebar, click Site Administration. You will see an array of tabs for all the tasks you can perform as a site administrator.



Monitoring Users

The Users tab on the admin dashboard displays the complete list of users. You can see which users are currently active, and when a user last logged in to Cloudera Machine Learning. To modify a user's username, email or permissions, click the Edit button under the Action column.

Synchronizing Users

You can synchronize Users within an ML Workspace with those that have been defined access at the Environment level (through the MLAdmin, MLUser, and MLBusinessUser roles). Doing so for new Users enables you to take administrative actions such as setting Team assignments, defining Project Collaborators, and more, all prior to the new Users' first time logging in to the Workspace.

To synchronize Users, go to Site Administration Users , and click Synchronize Users. This adds any users defined at the Environment level to the workspace, updates any role changes that have been made, and deactivates any users that have been deactivated.

Note: The Administrator should periodically perform user synchronization to ensure that users who are deactivated on the environment level are also deactivated in CML.

Related Information

Cloudera Machine Learning Email Notifications

Configuring Quotas

This topic describes how to configure CPU, GPU, and memory quotas for users of an ML workspace.

Before you begin

Required Role: MLAdmin

Note: On Private Cloud, the corresponding role is EnvironmentAdmin.

Make sure you are assigned the MLAdmin role in CDP. Only users with the MLAdmin role will be logged into ML workspaces with Site Administrator privileges.

There are two types of quota: Default and Custom. Default quotas apply to all users of the workspace. Custom quotas apply to individual users in the workspace, and take precedence over the default quota.

Procedure

1. Log in to the CDP web interface.
2. Click ML Workspaces, then open the Workspace for which you want to set quotas.
3. Click AdminQuotas.
4. Switch the Default Quotas toggle to ON.

   This applies a default quota of 2 vCPU and 8 GB memory to each user in the workspace.

   If your workspace was provisioned with GPUs, a default quota of 0 GPU per user applies. If you want users to have access to GPUs, you must modify the default quotas as described in the next step.
5. If you want to change the default quotas, click on Default (per user) .

   CML displays the Edit default quota dialog box.
6. Enter the CPU, Memory, and GPU quota values that should apply to all users of the workspace.
7. Click Update.
8. To add a custom quota for a specific user, click Add User.
9. Enter the user name, and enter the quotas for CPU, Memory, and GPU.

10. Click Add.

Results

Enabling and modifying quotas will only affect new workloads. If users have already scheduled workloads that exceed the new quota limits, those will continue to run uninterrupted. If a user is over their limit, they will not be able to schedule any more workloads.

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

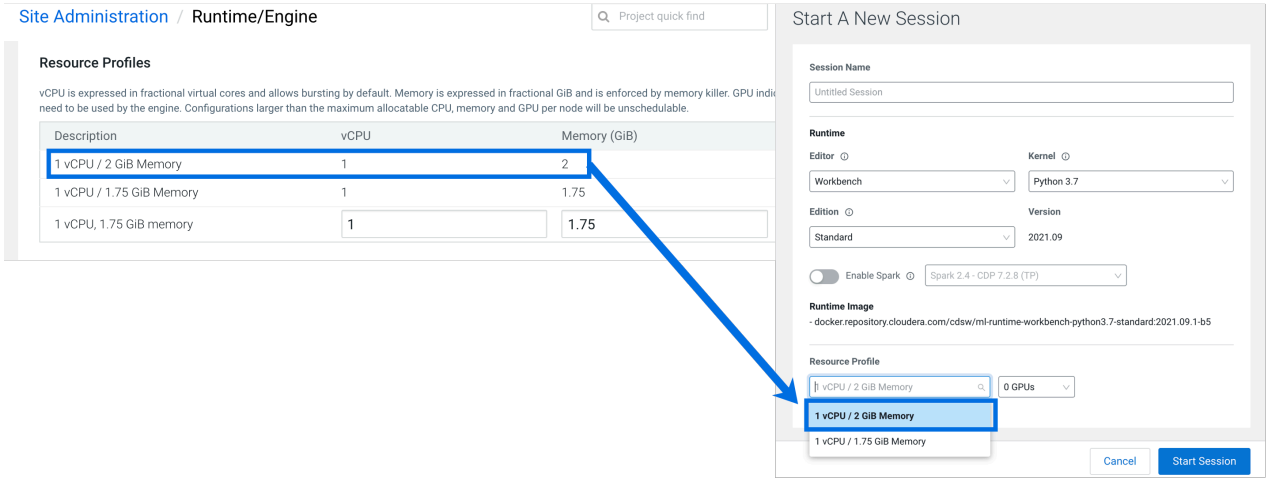
1. To create resource profiles, go to the Site Administration Runtime/Engine page.
2. Add a new profile under Resource Profiles.

   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 Cloudera Machine Learning hosts are equipped with GPUs, and you have enabled them for use by setting the relevant properties in cdsw.conf.

Results

Figure 11: Resource profiles available when launching a session



Onboarding Business Users

There are two procedures required for adding Business Users to CML. First, an Admin ensures the Business User has the correct permissions, and second, a Project Owner adds the Business User as a Collaborator.

Before you begin

Make sure the user is already assigned in your external identity provider, such as LDAP.

About this task

The Admin user performs these steps:

Procedure

1. In Environments, select the correct environment where the ML workspace is hosted.
2. In Manage Access, search for the user, and add the ML Business User role. Make sure the user does not already have a higher-level permission, such as ML Admin or ML User, either through a direct role assignment or a group membership.
3. Click Update Roles.
4. Inside the ML Workspace, go to Site Administration > Users , and click Synchronize Users. This adds the necessary Users defined at the Environment level to the Workspace, and updates any role changes that have been made.

What to do next

Add the ML Business User as a Collaborator to a Project.

Related Information

Adding a Collaborator

Adding a Collaborator

Project Owners can add Collaborators to a project.

About this task

The Project Owner performs these steps:

Procedure

1. Go to Collaborators, and enter the user id in the Search box.
2. Choose the user id, and click Add. The user is added with their role displayed.

Results

Now, when the Business User logs in, they are able to access the Applications under this project.

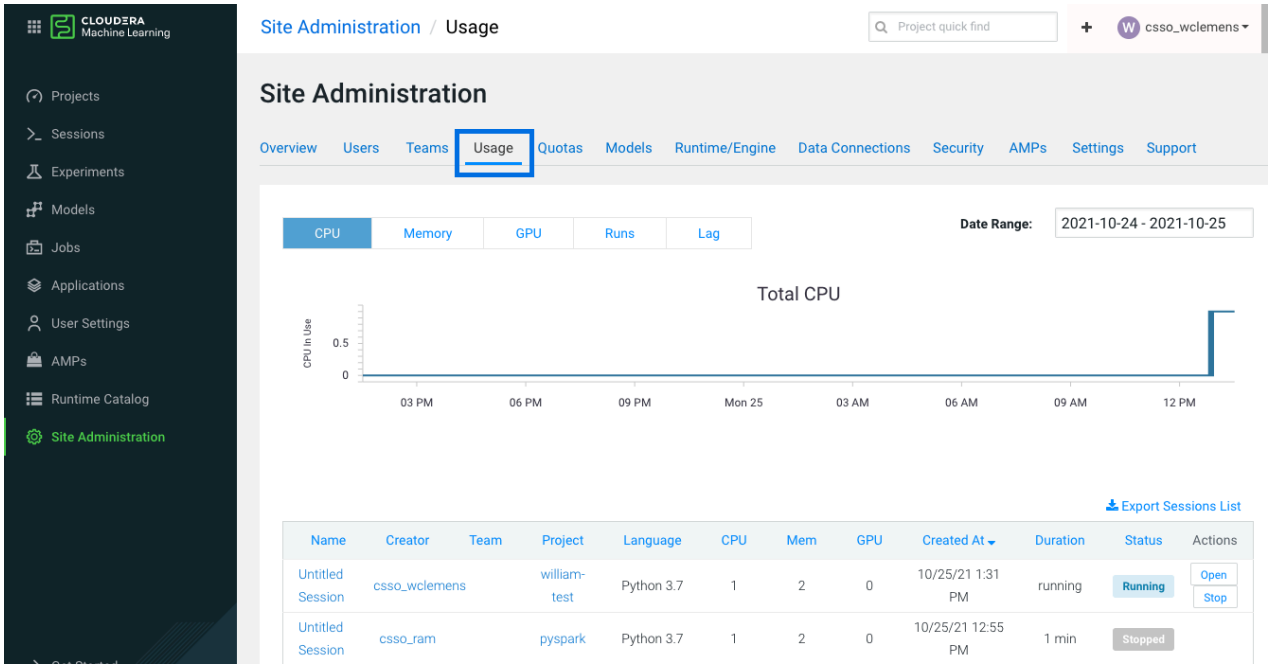
Monitoring Cloudera Machine Learning Activity

This topic describes how to monitor user activity on an ML workspace.

Required Role: Site Administrator

The Admin Overview tab displays basic information about your deployment, such as the number of users signed up, the number of teams and projects created, memory used, and some average job scheduling and run times. You can also see the version of Cloudera Machine Learning you are currently running.

The Admin Activity tab of the dashboard displays the following time series charts. These graphs should help site administrators identify basic usage patterns, understand how cluster resources are being utilized over time, and how they are being distributed among teams and users.



Important: The graphs and numbers on the Admin Activity page do not account for any resources used by active models on the deployment. For that information, go to Admin Models page.

- CPU - Total number of CPUs requested by sessions running at this time.

  Note that code running inside an n-CPU session, job, experiment or model replica can access at least n CPUs worth of CPU time. Each user pod can utilize all of its host's CPU resources except the amount requested by other user workloads or Cloudera Machine Learning application components. For example, a 1-core Python session can use more than 1 core if other cores have not been requested by other user workloads or CML application components.
- Memory - Total memory (in GiB) requested by sessions running at this time.
- GPU - Total number of GPUs requested by sessions running at this time.
- Runs - Total number of sessions and jobs running at this time.
- Lag - Depicts session scheduling and startup times.
  - Scheduling Duration: The amount of time it took for a session pod to be scheduled on the cluster.
  - Starting Duration: The amount of time it took for a session to be ready for user input. This is the amount of time since a pod was scheduled on the cluster until code could be executed.

The Export Sessions List provides a CSV export file of the columns listed in the table. It is important to note that the exported duration column is in seconds for a more detailed output.

Tracked User Events

The tables on this page describe the user events that are logged by Cloudera Machine Learning.

Table 21: Database Columns

When you query the user_events table, the following information can be returned:

Information	Description
id	The ID assigned to the event.
user_id	The UUID of the user who triggered the event.
ipaddr	The IP address of the user or component that triggered the event. 127.0.0.1 indicates an internal component.
user agent	The user agent for this action, such as the web browser. For example: `Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36`
event_name	The event that was logged. The tables on this page list possible events.
description	This field contains the model name and ID, the user type (NORMAL or ADMIN), and the username.
created_at	The date (YYYY-MM-DD format) and time (24-hour clock) the event occurred .

Table 22: Events Related to Engines

Event	Description
engine environment vars updated	-
engine mount created	-
engine mount deleted	-
engine mount updated	-
engine profile created	-
engine profile deleted	-
engine profile updated	-

Table 23: Events Related to Experiments

Event	Description
experiment run created	-
experiment run repeated	-
experiment run cancelled	-

Table 24: Events Related to Files

Event	Description
file downloaded	-
file updated	-
file deleted	-
file copied	-
file renamed	-
file linked	The logged event indicates when a symlink is created for a file or directory.
directory uploaded	-

Table 25: Events Related to Models

Event	Description
model created	-
model deleted	-

Table 26: Events Related to Jobs

Event	Description
job created	-
job started	-
stopped all runs for job	-
job shared with user	-
job unshared with user	-
job sharing updated	The logged event indicates when the sharing status for a job is changed from one of the following options to another: • All anonymous users with the link • All authenticated users with the link • Specific users and teams

Table 27: Events Related to Licenses

Event	Description
license created	-
license deleted	-

Table 28: Events Related to Projects

Event	Description
project created	-
project updated	-
project deleted	-
collaborator added	-
collaborator removed	-
collaborator invited	-

Table 29: Events Related to Sessions

Event	Description
session launched	-
session terminated	-
session stopped	-
session shared with user	-
session unshared with user	-

Event	Description
update session sharing status	The logged event indicates when the sharing status for a session is changed from one of the following options to another: • All anonymous users with the link • All authenticated users with the link • Specific users and teams

Table 30: Events Related to Admin Settings

site config updated	The logged event indicates when a setting on the Admin Settings page is changed.

Table 31: Events Related to Teams

Event	Description
add member to team	-
delete team member	-
update team member	-

Table 32: Events Related to Users

Event	Description
forgot password	-
password reset	-
update user	If the logged event shows that a user is banned, that means that the user account has been deactivated and does not count toward the license.
user signup	-
user login	The logged event includes the authorization method, LDAP/SAML or local.
user logout	-
ldap/saml user creation	The logged event indicates when a user is created with LDAP or SAML.

Monitoring User Events

This topic shows you how to query the PostgresSQL database that is embedded within the Cloudera Machine Learning deployment to monitor or audit user events.

About this task

Querying the PostgresSQL database that is embedded within the Cloudera Machine Learning deployment requires root access to the Cloudera Machine Learning Master host.

Procedure

1. SSH to the Cloudera Machine Learning Master host and log in as root.

   For example, the following command connects to cdsw-master-host. as root:

   ```
   ssh root@cdsw-master-host.yourcdswdomain.com
   ```

2. Get the name of the database pod:

```
kubectl get pods -l role=db
```

The command returns information similar to the following example:

```
NAME                  READY   STATUS    RESTARTS   AGE
db-86bbb69b54-d5q88   1/1     Running   0          4h46m
```

3. Enter the following command to log into the database as the sense user:

```
 kubectl exec <database pod> -ti -- psql -U sense
```

For example, the following command logs in to the database on pod db-86bbb69b54-d5q88:

```
kubectl exec db-86bbb69b54-d5q88 -ti -- psql -U sense
```

You are logged into the database as the sense user.

4. Run queries against the user_events table.

For example, run the following query to view the most recent user event:

```
select * from user_events order by created_at DESC LIMIT 1
```

The command returns information similar to the following:

```
id          | 3658
user_id     | 273
ipaddr      | ::ffff:127.0.0.1
user_agent  | node-superagent/2.3.0
event_name  | model created
description | {"model":"Simple Model 1559154287-ex5yn","modelId":"50","
userType":"NORMAL","username":"DonaldBatz"}
created_at  | 2019-05-29 18:24:47.65449
```

5. Optionally, you can export the user events to a CSV file for further analysis:
   a) Copy the user_events table to a CSV file:

      ```
      copy user_events to '/tmp/user_events.csv' DELIMITER ',' CSV HEADER;
      ```

   b) Find the container that the database runs on:

      ```
      docker ps | grep db-86bbb
      ```

      The command returns output similar to the following:

      ```
      c56d04bbd58 c230b2f564da "docker-entrypoint..." 7 days ago Up 7 days k8s
      _db_db-86bbb69b54-fcfm6_default_8b2dd23d-88b9-11e9-bc34-0245eb679f96_0
      ```

      The first entry is the container ID.
   c) Copy the user_events.csv file out of the container into a temporary directory on the Master host:

      ```
      docker cp <container ID>:/tmp/user_events.csv /tmp/user_events.csv
      ```

      For example:

      ```
      docker cp 8c56d04bbd58:/tmp/user_events.csv /tmp/user_events.csv
      ```

   d) Use SCP to copy /tmp/user_events.csv from the Cloudera Machine Learning Master host to a destination of your choice.

      For example, run the following command on your local machine to copy user_events.csv to a local directory named events:

      ```
      scp root@cdsw-master-host.yourcdswdomain.com:/tmp/user_events.csv /local
      /directory/events/
      ```

What to do next

For information about the different user events, see *Tracked User Events*.

Related Information

Tracked User Events

Monitoring Active Models Across the Workspace

This topic describes how to monitor all active models currently deployed on your workspace.

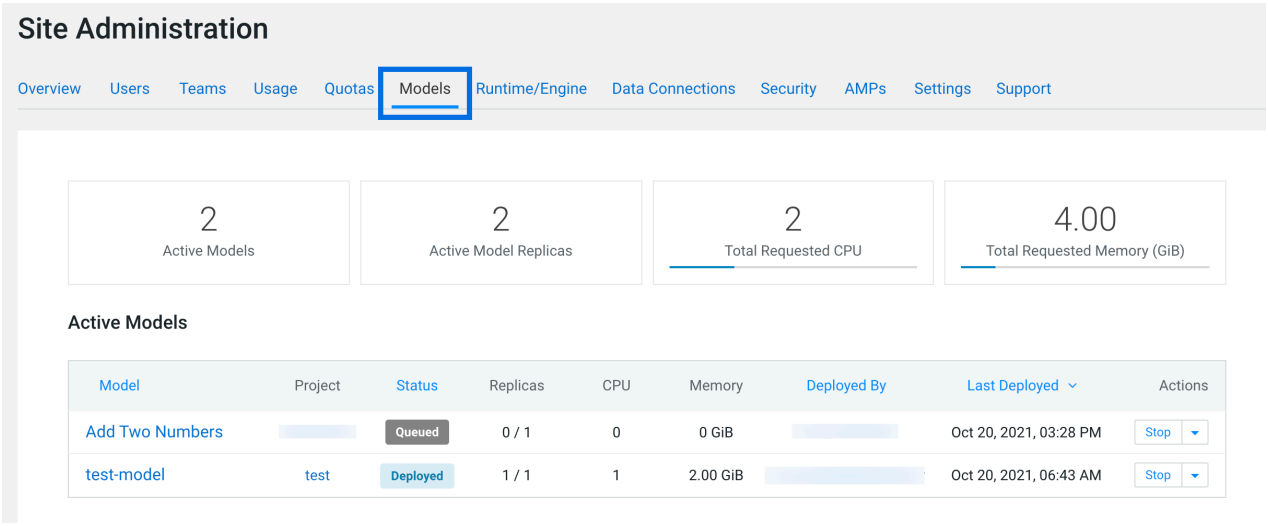
What is an Active Model?

A model that is in the Deploying, Deployed, or Stopping stages is referred to as an active model.

Monitoring All Active Models Across the Workspace

Required Role: Site Administrator

To see a complete list of all the models that have been deployed on a deployment, and review resource usage across the deployment by models alone, go to Admin Models . On this page, site administrators can also Stop/Restart/ Rebuild any of the currently deployed models.



Monitoring and Alerts

Cloudera Machine Learning leverages CDP Monitoring based on Prometheus and Grafana to provide dashboards that allow you to monitor how CPU, memory, storage, and other resources are being consumed by your ML workspaces.

Prometheus is an internal data source that is auto-populated with resource consumption data for each deployment. Grafana is the monitoring dashboard that allows you to create visualizations for resource consumption data from Prometheus. By default, CML provides three Grafana dashboards: K8 Cluster, K8s Containers, and K8s Node. You can extend these dashboards or create more panels for other metrics. For more information, see the Grafana documentation.

Related Information

Grafana documentation

Application Polling Endpoint

The CML server periodically polls applications for their status. The default polling endpoint is the root endpoint (/), but a custom polling endpoint can be specified if the server or other application has difficulty with the default endpoint.

When creating or modifying an application, you can specify a new value for the CDSW_APP_POLLING_END POINT environmental variable. Just replace the default value / that is shown. For more information, see *Analytical Applications*.

You can also set the environmental value in Project Settings Advanced . In this case, any setting made here can be overridden by settings in a given application. However, settings made in Project Settings Advanced also apply when polling sessions.

Related Information

Analytical Applications

Choosing Default Engine

This topic describes how to choose a default engine for creating projects.

Before you begin

Required Role: MLAdmin

Note: On Private Cloud, the corresponding role is EnvironmentAdmin.

Make sure you are assigned the MLAdmin role in CDP. Only users with the MLAdmin role will be logged into ML workspaces with Site Administrator privileges.

There are two types of default engines: ML Runtime and Legacy Engines. However, legacy engines are deprecated in the current release and project settings default to ML Runtime.

Legacy engines Engines contain the machinery necessary to run sessions using all four interpreter options that CML currently supports (Python 2, Python 3, R and Scala) and other support utilities (C and Fortran compilers, LaTeX, etc.). ML Runtimes are thinner and more lightweight than legacy engines. Rather than supporting multiple programming languages in a single engine, each Runtime variant supports a single interpreter version and a subset of utilities and libraries to run the user's code in Sessions, Jobs, Experiments, Models, or Applications.

Procedure

1. Log in to the CDP web interface.
2. Click ML Workspaces, then open the Workspace for which you want to set Default Engine.
3. Click Admin Runtime/Engine .
4. Choose the Default Engine you would like to use as the default for all newly created projects in this workspace.

   Note: Legacy Engines are deprecated in this release and Cloudera recommends using Runtime.

5. Modify the remaining information on the page:
   - Resource Profiles listed in the table are selectable resource options for both legacy Engines and ML Runtimes (for example, when starting a Session or Job)
   - The remaining information on the page applies to site-level settings specific for legacy Engines.

Related Information

ML Runtimes versus Legacy Engines

Controlling User Access to Features

Cloudera Machine Learning provides Site Administrators with the ability to restrict or hide specific functionality that non-Site Administrator users have access to in the UI. For example, a site administrator can hide the models and experiments features from the ML workspace UI.

The settings on this page can be configured through the Security and Settings tabs on the Admin page.

Table 33: Security Tab

Property	Description
Allow remote editing	Disable this property to prevent users from connecting to the Cloudera Machine Learning deployment with cdswctl and using local IDEs, such as PyCharm.
Allow only session creators to run commands on active sessions	By default, a user's permission to active sessions in a project is the same as the user's permission to that project, which is determined by the combination of the user's permission as a project collaborator, the user's permission in the team if this is a team project, and whether the user is a Site Administrator. By checking this checkbox, only the user that created the active session will be able to run commands in that session. No other users, regardless of their permissions in the team or as project collaborators, will be able to run commands on active sessions that are not created by them. Even Site Administrators will not be able to run commands in other users' active sessions.

Property	Description
Allow console output sharing	Disable this property to remove the Share button from the project workspace and workbench UI as well as disable access to all shared console outputs across the deployment. Note that re-enabling this property does not automatically grant access to previously shared consoles. You will need to manually share each console again
Allow anonymous access to shared console outputs	Disable this property to require users to be logged in to access shared console outputs.
Allow file upload/download through UI	Use this checkbox to show/hide file upload/download UI in the project workspace. When disabled, Cloudera Machine Learning API will forbid request of downloading file(s) as attachment. Note that the backend API to upload/edit/read the project files are intact regardless of this change in order to support basic Cloudera Machine Learning functionality such as file edit/read.

Table 34: Settings Tab

Property	Description
Require invitation to sign up	Enable this property to send email invitations to users when you add them to a group. To send email, an SMTP server must first be configured in Settings Email .
Allow users to create public projects	Disable this property to restrict users from creating new public projects. Site Administrators will have to create any new public projects.
Allow Legacy Engine users to use the Python 2 kernel	Enable this property to allow Legacy Engine users to select the Python 2 kernel when creating a job. Python 2 is disabled by default.
Allow users to create projects	Disable this property to restrict users from creating new projects. Site Administrators will have to create any new projects.
Allow users to create teams	Disable this property to restrict users from creating new teams. Site Administrators will have to create any new teams.
Allow users to run experiments	Disable this property to hide the Experiments feature in the UI. Note that this property does not affect any active experiments. It will also not stop any experiments that have already been queued for execution.
Allow users to create models	Disable this property to hide the Models feature in the UI. Note that this property does not affect any active models. In particular, if you do not stop active models before hiding the Models feature, they continue to serve requests and consume computing resources in the background.
Allow users to create applications	Disable this property to hide the Applications feature in the UI. Note that this property does not affect any active applications. In particular, if you do not stop active applications before hiding the feature, they continue to serve requests and consume computing resources in the background.

Cloudera Machine Learning Email Notifications

Cloudera Machine Learning allows you to send email notifications when you add collaborators to a project, share a project with a colleague, and for job status updates (email recipients are configured per-job). This topic shows you how to specify email address for such outbound communications.

Note that email notifications are not currently enabled by default. Emails are not sent when you create a new project. Email preferences cannot currently be configured at an individual user level.

Option 1: If your existing corporate SMTP server is accessible from the VPC where your ML workspace is running, you can continue to use that server. Go to the AdminSettings tab to specify an email address for outbound invitations and job notifications.

Option 2: If your existing SMTP solution cannot be used, consider using an email service provided by your cloud provider service. For example, Amazon provides Amazon Simple Email Service (Amazon SES).

Web session timeouts

You can set web sessions to time out and require the user to log in again. This time limit is not based on activity, it is the maximum time allowed for a web session.

You can set timeout limits for Users and Admin Users in Site AdministrationSecurity.

- User Web Browser Timeout (minutes) - This timeout sets the default maximum length of time that a web browser session can remain inactive. You remain logged in if you are actively using the session. If you are not active, then after a 5-minute warning, you are automatically logged out. Any changes to the setting take effect for any subsequent user logins.
- Admin User Web Browser Timeout (minutes) - This timeout sets the default maximum length of time that a web browser session for an Admin user can remain inactive. You remain logged in if you are actively using the session. If you are not active, then after a 5-minute warning, you are automatically logged out. Any changes to the setting take effect for any subsequent Admin user logins.

Project Garbage Collection

Marks orpaned files for deletion from a project and cleans up projects that are marked for deletion.

Procedure

1. Click Site Administration Settings .
2. Scroll to Project Garbage Collection.

   Click Garbage Collect Projects to permanently delete projects marked for deletion.

   Click Clean Up Orphaned Projects to mark orphaned projects for deletion.

Results

Orphaned project files are marked for deletion. All files marked for deletion are permanently deleted when you click Garbage Collect Projects.

How to make base cluster configuration changes

When you make base cluster configuration changes, you need to restart the base cluster to propagate those changes.

In general, as Administrator you perform the following steps:

1. Make the necessary configuration changes in the base cluster.
2. Restart the base cluster.
3. In the Private Cloud compute cluster, perform the specific Kubernetes commands below to restart the ds-cdh-client pods for CML.

For ECS:

1. Access Cloudera Manager.
2. Navigate to the Containerized Cluster ECS Web UI: Clusters Your embedded Cluster ECS Web UI ECS Web UI
3. Select the namespace of your ML Workspace on the top left dropdown.
4. Navigate to Workloads Deployments .
5. Locate ds-cdh-client in the list and perform Restart from the breadcrumbs on the right.

For OCP:

Access the openshift cluster with oc or kubectl, and scale the deployment of ds-cdh-client down and back up. Use the following commands.

1. oc scale deployment/ds-cdh-client --namespace <ml-namespace> --replicas 0
2. oc scale deployment/ds-cdh-client --namespace <ml-namespace> --replicas 1

Disable Addons

As a Private Cloud Administrator, you should ensure that Spark Runtime Addons used on your site that are compatible with the base cluster version. In practice, this means you should disable the incompatible versions that may be installed.

Note: Changing the default Hadoop CLI Runtime Addon causes jobs, models, and application workloads to be unable to start up. Please see the Release Notes for more information.

1. Go to the CDP management console, and determine the base cluster version.
2. In each CML workspace, in Site Administration Runtime ,
   - Set the Hadoop addon as default that has the base cluster version in its name.
   - Keep those Spark addons enabled that have the base cluster version in their name and disable other Spark Addons.
   - If some workloads have been configured to use a disabled Spark Addon, the affected workloads must be reconfigured to use an enabled Spark Addon. This can happen in the event a workspace is upgraded.

Configuring External Authentication with LDAP and SAML

Important: Cloudera recommends you leverage Single Sign-On for users via the CDP Management Console. For instructions on how to configure this, see Configuring LDAP authentication for CDP Private Cloud. If you cannot do this, we recommend contacting Cloudera Support before attempting to use the LDAP or SAML instructions provided in this section.

Cloudera Machine Learning supports user authentication against its internal local database, and against external services such as Active Directory, OpenLDAP-compatible directory services, and SAML 2.0 Identity Providers. By default, Cloudera Machine Learning performs user authentication against its internal local database. This topic describes the signup process for the first user, how to configure authentication using LDAP, Active Directory or SAML 2.0, and an optional workaround that allows site administrators to bypass external authentication by logging in using the local database in case of misconfiguration.

Configuring LDAP/Active Directory Authentication

This topic describes how to set up LDAP authentication for a workspace.

Important: This is not the recommended method to set up LDAP authentication. Cloudera recommends you use the CDP management console to set this up: Configuring LDAP authentication for CDP Private Cloud.

Cloudera Machine Learning supports both search bind and direct bind operations to authenticate against an LDAP or Active Directory directory service. The search bind authentication mechanism performs an ldapsearch against the directory service, and binds using the found Distinguished Name (DN) and password provided. The direct bind authentication mechanism binds to the LDAP server using a username and password provided at login.

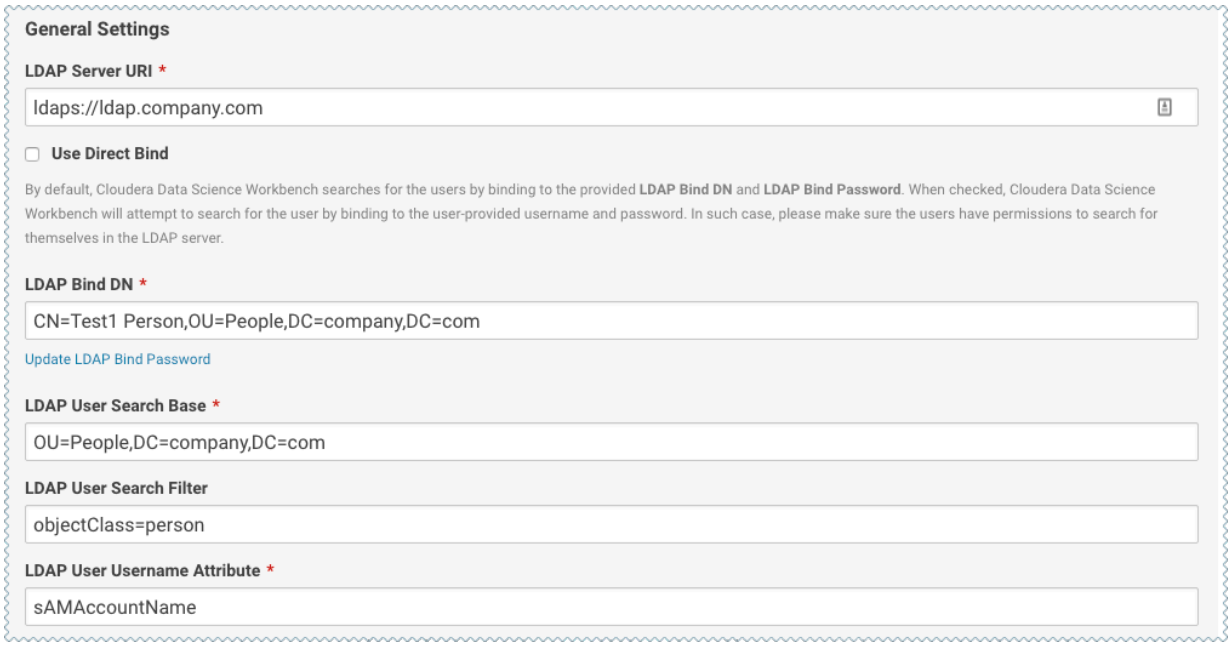
You can configure Cloudera Machine Learning to use external authentication methods by clicking the Admin link on the left sidebar and selecting the Security tab. Select LDAP from the list to start configuring LDAP properties.

LDAP General Settings

Lists the general settings required to configure LDAP authentication.

- LDAP Server URI: Required. The URI of the LDAP/Active Directory server against which Cloudera Machine Learning should authenticate. For example, ldaps://ldap.*company*.com:636.

- Use Direct Bind: If checked, the username and password provided at login are used with the LDAP Username Pattern for binding to the LDAP server. If unchecked, Cloudera Machine Learning uses the search bind mechanism and two configurations, LDAP Bind DN and LDAP Bind Password, are required to perform the ldapsearch against the LDAP server.
- LDAP Bind DN: Required when using search bind. The DN to bind to for performing ldapsearch. For example, cn=admin,dc=company,dc=com.
- LDAP Bind Password: Required when using search bind. This is the password for the LDAP Bind DN.
- LDAP Search Base: Required. The base DN from which to search for the provided LDAP credentials. For example, ou=Engineering,dc=company,dc=com.
- LDAP User Filter: Required. The LDAP filter for searching for users. For example, (&(sAMAccountName={0})(objectclass=person)). The {0} placeholder will be replaced with the username provided at login.
- LDAP User Username Attribute: Required. The case-sensitive username attribute of the LDAP directory service. This is used by Cloudera Machine Learning to perform the bind operation and extract the username from the response. Common values are uid, sAMAccountName, or userPrincipalName.



When you select Use Direct Bind, Cloudera Machine Learning performs a direct bind to the LDAP server using the LDAP Username Pattern with the credentials provided on login (not LDAP Bind DN and LDAP Bind Password).

By default, Cloudera Machine Learning performs an LDAP search using the bind DN and credentials specified for the LDAP Bind DN and LDAP Bind Password configurations. It searches the subtree, starting from the base DN specified for the LDAP Search Base field, for an entry whose attribute specified in LDAP User Username Attribute, has the same value as the username provided on login. Cloudera Machine Learning then validates the user-provided password against the DN found as a result of the search.

LDAP Group Settings

In addition to the general LDAP settings, you can use group settings to restrict the access to Cloudera Machine Learning to certain groups in LDAP.

- LDAP Group Search Base: The base distinguished name (DN) where Cloudera Machine Learning will search for groups.
- LDAP Group Search Filter: The LDAP filter that Cloudera Machine Learning will use to determine whether a user is affiliated to a group.

  A group object in LDAP or Active Directory typically has one or more member attributes that stores the DNs of users in the group. If LDAP Group Search Filter is set to member={0}, Cloudera Machine Learning will automatically substitute the {0} placeholder for the DN of the authenticated user.

- LDAP User Groups: A list of LDAP groups whose users have access to Cloudera Machine Learning. When this property is set, only users that successfully authenticate themselves AND are affiliated to at least one of the groups listed here, will be able to access Cloudera Machine Learning.

  If this property is left empty, all users that can successfully authenticate themselves to LDAP will be able to access Cloudera Machine Learning.
- LDAP Full Administrator Groups: A list of LDAP groups whose users are automatically granted the site administrator role on Cloudera Machine Learning.

  The groups listed under LDAP Full Administrator Groups do not need to be listed again under the LDAP User Groups property.

  Figure 12: Example

  If you want to restrict access to Cloudera Machine Learning to members of a group whose DN is:

  ```
  CN=CMLUsers,OU=Groups,DC=company,DC=com
  ```

  And automatically grant site administrator privileges to members of a group whose DN is:

  ```
  CN=CMLAdmins,OU=Groups,DC=company,DC=com
  ```

  Add the CNs of both groups to the following settings in Cloudera Machine Learning:

  - LDAP User Groups: CMLUsers
  - LDAP Full Administrator Groups: CMLAdmins

Test LDAP Configuration

Use the Test LDAP Configuration form to test your settings.

You can test your LDAP/Active Directory configuration by entering your username and password in the Test LDAP Configuration section. This form simulates the user login process and allows you to verify the validity of your LDAP/Active Directory configuration without opening a new window.

Before using this form, make sure you click Update to save the LDAP configuration you want to test.

Configuring SAML Authentication

This topic describes how to set up SAML for Single Sign-on authentication for a workspace.

Important: This is not the recommended method to set up SSO. Cloudera recommends you use the CDP management console to set this up: Configuring LDAP authentication for CDP Private Cloud.

Cloudera Machine Learning supports the Security Assertion Markup Language (SAML) for Single Sign-on (SSO) authentication; in particular, between an identity provider (IDP) and a service provider (SP). The SAML specification defines three roles: the principal (typically a user), the IDP, and the SP. In the use case addressed by SAML, the principal (user agent) requests a service from the service provider. The service provider requests and obtains an identity assertion from the IDP. On the basis of this assertion, the SP can make an access control decision—in other words it can decide whether to perform some service for the connected principal.

The primary SAML use case is called web browser single sign-on (SSO). A user with a user agent (usually a web browser) requests a web resource protected by a SAML SP. The SP, wanting to know the identity of the requesting user, issues an authentication request to a SAML IDP through the user agent. In the context of this terminology, Cloudera Machine Learning operates as a SP.

Cloudera Machine Learning supports both SP- and IDP-initiated SAML 2.0-based SSO. Its Assertion Consumer Service (ACS) API endpoint is for consuming assertions received from the Identity Provider. If your Cloudera Machine Learning domain root were cdsw.*company*.com, then this endpoint would be available at http://cdsw.*company*.com/api/v1/saml/acs. SAML 2.0 metadata is available at http://cdsw.*company*.com/api/v1/saml/metadata for IDP-initiated SSO. Cloudera Machine Learning uses HTTP Redirect Binding for authentication requests and expects to receive responses from HTTP POST Binding.

When Cloudera Machine Learning receives the SAML responses from the Identity Provider, it expects to see at least the following user attributes in the SAML responses:

- The unique identifier or username. Valid attributes are:
  - uid
  - urn:oid:0.9.2342.19200300.100.1.1
- The email address. Valid attributes are:
  - mail
  - email
  - urn:oid:0.9.2342.19200300.100.1.3
- The common name or full name of the user. Valid attributes are:
  - cn
  - urn:oid:2.5.4.3

  In the absence of the cn attribute, Cloudera Machine Learning will attempt to use the following user attributes, if they exist, as the full name of the user:
  - The first name of the user. Valid attributes are:
    - givenName
    - urn:oid:2.5.4.42
  - The last name of the user. Valid attributes are:
    - sn
    - urn:oid:2.5.4.4

Configuration Options

List of properties to configure SAML authentication and authorization in Cloudera Machine Learning.

Cloudera Machine Learning Settings

- Entity ID: Required. A globally unique name for Cloudera Machine Learning as a Service Provider. This is typically the URI.
- NameID Format: Optional. The name identifier format for both Cloudera Machine Learning and Identity Provider to communicate with each other regarding a user. Default: urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress.
- Authentication Context: Optional. SAML authentication context classes are URIs that specify authentication methods used in SAML authentication requests and authentication statements. Default: urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport.

Signing SAML Authentication Requests

- CDSW Private Key for Signing Authentication Requests: Optional. If you upload a private key, you must upload a corresponding certificate as well so that the Identity Provider can use the certificate to verify the authentication requests sent by Cloudera Machine Learning. You can upload the private key used for both signing authentication requests sent to Identity Provider and decrypting assertions received from the Identity Provider.
- CML Certificate for Signature Validation: Required if the Cloudera Machine Learning Private Key is set, otherwise optional. You can upload a certificate in the PEM format for the Identity Provider to verify the authenticity of the authentication requests generated by Cloudera Machine Learning. The uploaded certificate is made available at the http://cdsw.*company*.com/api/v1/saml/metadata endpoint.

SAML Assertion Decryption

Cloudera Machine Learning uses the following properties to support SAML assertion encryption & decryption.

- CML Certificate for Encrypting SAML Assertions - Must be configured on the Identity Provider so that Identity Provider can use it for encrypting SAML assertions for Cloudera Machine Learning

- CML Private Key for Decrypting SAML Assertions - Used to decrypt the encrypted SAML assertions.

Identity Provider

- Identity Provider SSO URL: Required. The entry point of the Identity Provider in the form of URI.
- Identity Provider Signing Certificate: Optional. Administrators can upload the X.509 certificate of the Identity Provider for Cloudera Machine Learning to validate the incoming SAML responses.

  Cloudera Machine Learning extracts the Identity Provider SSO URL and Identity Provider Signing Certificate information from the uploaded Identity Provider Metadata file. Cloudera Machine Learning also expects all Identity Provider metadata to be defined in a <md:EntityDescriptor> XML element with the namespace "urn:oasis:names:tc:SAML:2.0:metadata", as defined in the SAMLMeta-xsd schema.

  For on-premises deployments, you must provide a certificate and private key, generated and signed with your trusted Certificate Authority, for Cloudera Machine Learning to establish secure communication with the Identity Provider.

Authorization

When you're using SAML 2.0 authentication, you can use the following properties to restrict the access to Cloudera Machine Learning to certain groups of users:

- SAML Attribute Identifier for User Role: The Object Identifier (OID) of the user attribute that will be provided by your identity provider for identifying a user's role/affiliation. You can use this field in combination with the following SAML User Groups property to restrict access to Cloudera Machine Learning to only members of certain groups.

  For example, if your identity provider returns the OrganizationalUnitName user attribute, you would specify the OID of the OrganizationalUnitName, which is urn:oid:2.5.4.11, as the value for this property.
- SAML User Groups: A list of groups whose users have access to Cloudera Machine Learning. When this property is set, only users that are successfully authenticated AND are affiliated to at least one of the groups listed here, will be able to access Cloudera Machine Learning.

  For example, if your identity provider returns the OrganizationalUnitName user attribute, add the value of this attribute to the SAML User Groups list to restrict access to Cloudera Machine Learning to that group.

  If this property is left empty, all users that can successfully authenticate themselves will be able to access Cloudera Machine Learning.
- SAML Full Administrator Groups: A list of groups whose users are automatically granted the site administrator role on Cloudera Machine Learning.

  The groups listed under SAML Full Administrator Groups do not need to be listed again under the SAML User Groups property.

Configuring HTTP Headers for Cloudera Machine Learning

This topic explains how to customize the HTTP headers that are accepted by Cloudera Machine Learning.

Required Role: Site Administrator

These properties are available under the site administrator panel at Admin Security .

Important: Any changes to the following properties require a full restart of Cloudera Machine Learning. To do so, run cdsw restart on the master host.

Enable Cross-Origin Resource Sharing (CORS)

Most modern browsers implement the Same-Origin Policy, which restricts how a document or a script loaded from one origin can interact with a resource from another origin. When the Enable cross-origin resource sharing property

is enabled on Cloudera Machine Learning, web servers will include the Access-Control-Allow-Origin: * HTTP header in their HTTP responses. This gives web applications on different domains permission to access the Cloudera Machine Learning API through browsers.

This property is disabled by default .

If this property is disabled, web applications from different domains will not be able to programmatically communicate with the Cloudera Machine Learning API through browsers.

Enable HTTP Security Headers

When Enable HTTP security headers is enabled, the following HTTP headers will be included in HTTP responses from servers:

- X-XSS-Protection
- X-DNS-Prefetch-Control
- X-Frame-Options
- X-Download-Options
- X-Content-Type-Options

This property is enabled by default .

Disabling this property could leave your Cloudera Machine Learning deployment vulnerable to clickjacking, cross-site scripting (XSS), or any other injection attacks.

Enable HTTP Strict Transport Security (HSTS)

Note: Without TLS/SSL enabled, configuring this property will have no effect on your browser.

When both TLS/SSL and this property (Enable HTTP Strict Transport Security (HSTS)) are enabled, Cloudera Machine Learning will inform your browser that it should never load the site using HTTP. Additionally, all attempts to access Cloudera Machine Learning using HTTP will automatically be converted to HTTPS.

This property is disabled by default .

If you ever need to downgrade to back to HTTP, use the following sequence of steps: First, deactivate this checkbox to disable HSTS and restart Cloudera Machine Learning. Then, load the Cloudera Machine Learning web application in each browser to clear the respective browser's HSTS setting. Finally, disable TLS/SSL across the cluster. Following this sequence should help avoid a situation where users get locked out of their accounts due to browser caching.

Enable HTTP Security Headers

When Enable HTTP security headers is enabled, the following HTTP headers will be included in HTTP responses from servers:

- X-XSS-Protection
- X-DNS-Prefetch-Control
- X-Frame-Options
- X-Download-Options
- X-Content-Type-Options

This property is enabled by default .

Disabling this property could leave your Cloudera Machine Learning deployment vulnerable to clickjacking, cross-site scripting (XSS), or any other injection attacks.

Enable HTTP Strict Transport Security (HSTS)

Note: Without TLS/SSL enabled, configuring this property will have no effect on your browser.

When both TLS/SSL and this property (Enable HTTP Strict Transport Security (HSTS)) are enabled, Cloudera Machine Learning will inform your browser that it should never load the site using HTTP. Additionally, all attempts to access Cloudera Machine Learning using HTTP will automatically be converted to HTTPS.

This property is disabled by default .

If you ever need to downgrade to back to HTTP, use the following sequence of steps: First, deactivate this checkbox to disable HSTS and restart Cloudera Machine Learning. Then, load the Cloudera Machine Learning web application in each browser to clear the respective browser's HSTS setting. Finally, disable TLS/SSL across the cluster. Following this sequence should help avoid a situation where users get locked out of their accounts due to browser caching.

Enable Cross-Origin Resource Sharing (CORS)

Most modern browsers implement the Same-Origin Policy, which restricts how a document or a script loaded from one origin can interact with a resource from another origin. When the Enable cross-origin resource sharing property is enabled on Cloudera Machine Learning, web servers will include the Access-Control-Allow-Origin: * HTTP header in their HTTP responses. This gives web applications on different domains permission to access the Cloudera Machine Learning API through browsers.

This property is disabled by default .

If this property is disabled, web applications from different domains will not be able to programmatically communicate with the Cloudera Machine Learning API through browsers.

SSH Keys

This topic describes the different types of SSH keys used by Cloudera Machine Learning, and how you can use those keys to authenticate to an external service such as GitHub.

Personal Key

Cloudera Machine Learning automatically generates an SSH key pair for your user account. You can rotate the key pair and view your public key on your user settings page. It is not possible for anyone to view your private key.

Every console you run has your account's private key loaded into its SSH-agent. Your consoles can use the private key to authenticate to external services, such as GitHub. For instructions, see #unique_229.

Team Key

Team SSH keys provide a useful way to give an entire team access to external resources such as databases or GitHub repositories (as described in the next section).

Like Cloudera Machine Learning users, each Cloudera Machine Learning team has an associated SSH key. You can access the public key from the team's account settings. Click Account, then select the team from the drop-down menu at the upper right corner of the page.

When you launch a console in a project owned by a team, you can use that team's SSH key from within the console.

Adding an SSH Key to GitHub

Cloudera Machine Learning creates a public SSH key for each account. You can add this SSH public key to your GitHub account if you want to use password-protected GitHub repositories to create new projects or collaborate on projects.

Procedure

1. Sign in to Cloudera Machine Learning.

2. Go to the upper right drop-down menu and switch context to the account whose key you want to add. This could be a individual user account or a team account.
3. On the left sidebar, click User Settings.
4. Go to the Outbound SSH tab and copy the User Public SSH Key.
5. Sign in to your GitHub account and add the Cloudera Machine Learning key copied in the previous step to your GitHub account. For instructions, refer the GitHub documentation on Adding a new SSH key to your GitHub account.

Creating an SSH Tunnel

You can use your SSH key to connect Cloudera Machine Learning to an external database or cluster by creating an SSH tunnel.

About this task

In some environments, external databases and data sources reside behind restrictive firewalls. A common pattern is to provide access to these services using a bastion host with only the SSH port open. Cloudera Machine Learning provides a convenient way to connect to such resources using an SSH tunnel.

If you create an SSH tunnel to an external server in one of your projects, then all engines that you run in that project are able to connect securely to a port on that server by connecting to a local port. The encrypted tunnel is completely transparent to the user and code.

Procedure

1. Open the Project Settings page.
2. Open the Tunnels tab.
3. Click New Tunnel.
4. Enter the server IP Address or DNS hostname.
5. Enter your username on the server.
6. Enter the local port that should be proxied, and to which remote port on the server.

What to do next

On the remote server, configure SSH to accept password-less logins using your individual or team SSH key. Often, you can do so by appending the SSH key to the file /home/username/.ssh/authorized_keys.

Autoscaling Workloads with Kubernetes

Autoscaling on Private Cloud

CML on Private Cloud supports application autoscaling on multiple fronts. Additional compute resources are utilized when users self-provision sessions, run jobs, and utilize other compute capabilities. Within a session, users can also leverage the worker API to launch resources necessary to host TensorFlow, PyTorch, or other distributed applications. Spark on Kubernetes scales up to any number of executors as requested by the user at runtime.

Restricting User-Controlled Kubernetes Pods

Cloudera Machine Learning includes three properties that allow you to control the permissions granted to user-controlled Kubernetes pods.

Required Role: Site Administrator

An example of a user-controlled pod is the engine pod, which provides the environment for sessions, jobs, etc. These pods are launched in a per-user Kubernetes namespace. Since the user has the ability to launch arbitrary pods, these settings restrict what those pods can do.

They are available under the site administrator panel at Admin Security under the Control of User-Created Kubernetes Pods section.

Do not modify these settings unless you need to run pods that require special privileges. Enabling any of these properties puts CML user data at risk.

Allow privileged pod containers

Pod containers that are "privileged" are extraordinarily powerful. Processes within such containers get almost the same privileges that are available to processes outside the container.

If this property is enabled, a privileged container could potentially access all data on the host.

This property is disabled by default .

Allow pod containers to mount unsupported volume types

The volumes that can be mounted inside a container in a Kubernetes pod are already heavily restricted. Access is normally denied to volume types that are unfamiliar, such as GlusterFS, Cinder, Fibre Channel, etc. If this property is enabled, pods will be able to mount all unsupported volume types.

This property is disabled by default .

Hadoop Authentication for ML Workspaces

About this task

CML does not assume that your Kerberos principal is always the same as your login information. Therefore, you will need to make sure CML knows your Kerberos identity when you sign in.

This procedure is required if you want to run Spark workloads in an ML workspace.

Procedure

1. Navigate to your ML workspace.
2. Go to the top-right dropdown menu, click Account settings Hadoop Authentication .
3. To authenticate, either enter your password or click Upload Keytab to upload the keytab file directly.

Results

Once successfully authenticated, Cloudera Machine Learning uses your stored credentials to ensure you are secure when running workloads.

CML and outbound network access

Cloudera Machine Learning expects access to certain external networks. See the related information *Configuring proxy hosts for CML workspace connections* for further information.

Note: The outbound network access destinations listed in *Configuring proxy hosts for CML workspace connections* are only the minimal set required for CDP installation and operation. For environments with limited outbound internet access due to using a firewall or proxy, access to Python or R package repositories such as Python Package Index or CRAN may need to be whitelisted if your use cases require installing packages from those repositories. Alternatively, you may consider creating mirrors of those repositories within your environment.

Related Information

Configuring proxy hosts for CML workspace connections

Troubleshooting

Troubleshooting tips may help you out of some situations with Cloudera Machine Learning.

Troubleshooting

This topic describes a recommended series of steps to help you start diagnosing issues with a Cloudera Machine Learning workspace.

- Issues with Provisioning ML Workspaces: If provisioning an ML workspace fails, make sure that you have all the resources required to provision an ML workspace. If failures persist, start debugging by reviewing the error messages on the screen. Check the workspace logs to see what went wrong.
- Issues with Accessing ML Workspaces: If your ML Admin has already provisioned a workspace for you but attempting to access the workspace fails, confirm with your ML Admin that they have completed all the steps required to grant you access.
- Issues with Running Workloads: If you have access to a workspace but are having trouble running sessions/jobs/experiments, and so on, see if your error is already listed here: Troubleshooting Issues with Workloads on page 215.

Cloudera Support

If you need assistance, contact Cloudera Support. Cloudera customers can register for an account to create a support ticket at the support portal. For CDP issues in particular, make sure you include the Request ID associated with your error message in the support case you create.

Downloading diagnostic bundles for a workspace

Learn how to manage and download diagnostic bundles.

The CDP platform provides various services for managing and downloading diagnostic bundles.

For more information, see: Sending Usage and Diagnostic Data to Cloudera

Troubleshooting Issues with Workloads

This section describes some potential issues data scientists might encounter once the ML workspace is running workloads.

401 Error caused by incompatible Data Lake version

The following error might occur due to an incompatible Data Lake version:

```
org.apache.ranger.raz.hook.s3.RazS3ClientCredentialsException: Exception in
Raz Server;
Check the raz server logs for more details, HttpStatus: 401
```

To avoid this issue, ensure that:

- Data Lake and Runtime (server) version is 7.2.11 or higher.
- Hadoop Runtime add-on (client) used in the CML session is 7.2.11 or higher.
- Spark Runtime add-on version must be CDE 1.13 or higher.

Engines cannot be scheduled due to lack of CPU or memory

A symptom of this is the following error message in the Workbench: "Unschedulable: No node in the cluster currently has enough CPU or memory to run the engine."

Either shut down some running sessions or jobs or provision more hosts for Cloudera Machine Learning.

Workbench prompt flashes red and does not take input

The Workbench prompt flashing red indicates that the session is not currently ready to take input.

Cloudera Machine Learning does not currently support non-REPL interaction. One workaround is to skip the prompt using appropriate command-line arguments. Otherwise, consider using the terminal to answer interactive prompts.

PySpark jobs fail due to Python version mismatch

```
Exception: Python in worker has different version 2.6 than that in driver 2.
7, PySpark cannot run with different minor versions
```

One solution is to install the matching Python 2.7 version on all the cluster hosts. A better solution is to install the Anaconda parcel on all CDH cluster hosts. Cloudera Machine Learning Python engines will use the version of Python included in the Anaconda parcel which ensures Python versions between driver and workers will always match. Any library paths in workloads sent from drivers to workers will also match because Anaconda is present in the same location across all hosts. Once the parcel has been installed, set the PYSPARK_PYTHON environment variable in the Cloudera Machine Learning Admin dashboard.

Troubleshooting Kerberos Errors

This topic describes some common Kerberos issues and their recommended solutions.

HDFS commands fail with Kerberos errors even though Kerberos authentication is successful in the web application

If Kerberos authentication is successful in the web application, and the output of klist in the engine reveals a valid-looking TGT, but commands such as hdfs dfs -ls / still fail with a Kerberos error, it is possible that your cluster is missing the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File. The JCE policy file is required when Red Hat uses AES-256 encryption. This library should be installed on each cluster host and will live under $JAVA_HOME. For more information, see Using AES-256 Encryption.

Cannot find renewable Kerberos TGT

Cloudera Machine Learning runs its own Kerberos TGT renewer which produces non-renewable TGT. However, this confuses Hadoop's renewer which looks for renewable TGTs. If the Spark 2 logging level is set to WARN or lower, you may see exceptions such as:

```
16/12/24 16:38:40 WARN security.UserGroupInformation: Exception encountered
while running the renewal command. Aborting renew thread. ExitCodeException
exitCode=1: kinit: Resource temporarily unavailable while renewing credentia
ls

16/12/24 16:41:23 WARN security.UserGroupInformation: PriviledgedActionExcep
tion as:user@CLOUDERA.LOCAL (auth:KERBEROS) cause:javax.security.sasl.SaslEx
ception: GSS initiate failed [Caused by GSSException: No valid credentials p
rovided (Mechanism level: Failed to find any Kerberos tgt)]
```

This is not a bug. Spark 2 workloads will not be affected by this. Access to Kerberized resources should also work as expected.

Reference

CML API v2

Cloudera Machine Learning exposes a REST API that you can use to perform operations related to projects, jobs, and runs. You can use API commands to integrate CML with third-party workflow tools or to control CML from the command line.

API v2 supersedes the existing Jobs API. For more information on the Jobs API, see Jobs API in the Related information section, below.

How to view the API Specification

You can view the comprehensive API specification on the REST API v2 Reference page. See Related information, below, for the link.

You can also obtain the specification of the available API commands directly from CML. In a browser, enter the following addresses:

- REST API: https://<domain name of CML instance>/api/v2/swagger.html
- Python API: https://<domain name of CML instance>/api/v2/python.html

You can also get json formatted output, by specifying swagger.json.

Quickstart

API key authentication

To get started, generate an API key. The API key is a randomly generated token that is unique to each user. It must be treated as highly sensitive information because it can be used to start jobs via the API. You need this API key to use in API calls.

1. Sign in to Cloudera Machine Learning.
2. In User Settings API Keys , click Create API Key.
3. Copy this API key to the clipboard.

Using `curl` from the command line

To use the `curl` command, it is convenient to store the domain and API key in environmental variables, as shown here:

1. Copy the API key.
2. Open a terminal, and store it to a variable. On unix-based systems:

   export API_KEY=<paste the API key value here>
3. Copy the domain name, which is in the address bar of the browser. On unix-based systems: export CDSW_DOMAIN=<domain> (a value like: ml-xxxx123456.com).

Example commands

If you have some projects, jobs, and runs already set up in your ML workspace, here are some commands to try:

- List available projects:

  ```
  curl -X GET -H "authorization: Bearer $API_KEY" https://$CDSW_DOMAIN/api/
  v2/projects | jq
  ```

  You can format the output for readability by piping through jq, a formatting utility.

- You can filter the output like so:

  ```
  curl -X GET -H “authorization: Bearer $API_KEY” https://$CDSW_DOMAIN/api/
  v2/projects?searchFilter=demo |  jq
  ```

  The output is limited to those projects that have the word “demo” in them.

  You can also paginate the output, for example by limiting each page to two projects. To do this, replace the string starting from the ‘?’ character with this: ?pageSize=2

  The output ends with a `next_page_token` and a string value. To get the next page use this: ?pageSize=2& pageToken=<token>

  Note: You have to add quotes around the entire https string because of the ampersand (&) character.

Using the Python client library

To use the Python API in your own code, first install the Python API client and point it to your cluster.

```
pip3 install https://$CDSW_DOMAIN/api/v2/python.tar.gz
```

Include the following code, and specify the values for <CDSW_DOMAIN> and <API_KEY> with variables or values for your installation.

```
# In a session:
    api_instance = default_client()
   # Outside a session:
   default_client("https://"+cluster, APIKEY)
```

Note: If you use default_client() in a session, no arguments are needed. If you use it outside of a session, you must provide the cluster name and API v2 key.

Then you can use commands in your script, such as a call to list projects:

```
projects = api_instance.list_projects()
```

The API returns objects that store values as attributes. To access the values, use dot notation. Do not use bracket notation as you would with a dictionary. For example:

```
myproj = client.create_project(...)
# This doesn't work:
myproj["id"]

# But this does
myproj.id
```

Check the Python documentation for a full list of the available API commands.

Using the Python client library in the REPL

Here is an example of a stub Python script that contains the environmental variables for your installation. Save it to your computer and run it locally to get a Python prompt for API commands.

demo.py

```
import clap
import argparse
```

```
parser = argparse.ArgumentParser(description='Test the generated python pac
kage.')
parser.add_argument("—host", type=str, help='The host name of your workspace
")
parser.add_argument("—token", type=str, help='Your API key")
args = parser.parse_args()

config = clap.Configuration()
config.host = ars.host
client = cmlapi.ApiClient(config)
client.set_default_header("authorization", "Bearer " + args.token)
api = cmlapi.Apiapi(client)
```

Run the script from your command line:

```
python3 -i demo.py —host https://$CDSW_DOMAIN —token $API_KEY
```

This returns a Python prompt with api available. You can run api calls from the prompt, as follows:

```
>>> api
<cmlapi.api.api_api.ApiApi object at 0xlasjoid>
>>> api.api_list_projects()
```

You can specify a search filter, such as the following:

```
api.api_list_projects(searchFilter='demo')

api.api_list_projects(page_size=2)

api.api_list_projects(page_size=2, page_token='<token value>')
```

Related Information

Cloudera Machine Learning REST API v2 Reference

Cloudera Data Science Workbench API v2

API v2 Usage

You can use API v2 to perform actions on Projects, Jobs, Models, and Applications.

Set up the client

The client is the object you use for executing API commands. You obtain the client from the CML cluster using your API key and the default_client() function.

Start by downloading the Python library directly from the workspace:

```
> pip3 install <workspace domain>/api/v2/python.tar.gz
```

Next, get an API key. Go to User Settings API Keys . Select Create API Key. Copy the generated API Key value to the clipboard.

Create an instance of the API:

```
> import cmlapi
> client = cmlapi.default_client(url="<workspace domain>", cml_api_key="<api
 key>")
> client.list_projects()
```

If your workspace is using a custom self-signed certificate, you might need to include it when creating the client:

```
> config = cmlapi.Configuration()
> config.host = “<workspace domain>”
> config.ssl_ca_cert = "<path to SSL certificate>"
> api = cmlapi.ApiClient(config)
> api.set_default_header("authorization", "Bearer " + “<api key>”)
> client = cmlapi.CMLServiceApi(api)
```

Using the Project API

To list the available projects:

```
projects = client.list_projects() # returns the first 10
second_page_projects = client.list_projects(page_token=projects.next_page_t
oken) # returns the next 10
lots_of_projects = client.list_projects(page_size=100)
second_page_lots_of_projects = client.list_projects(page_size=100, page_t
oken=lots_of_projects.next_page_token) # must re-include the same page size
for future pages
filtered_projects = client.list_projects(search_filter=”production”) # retu
rns all projects with “production” in the name or description
```

Select a particular project ID, and then validate it with a command to fetch the project:

```
project = client.get_project(project_id=”<project id>”)
```

Delete a project with the following command:

```
client.delete_project(project_id=”<project id>”)
```

Using the Jobs API

You can list out the jobs in the project like so:

```
jobs = client.list_jobs(project_id=”<projectid>”)
```

The same searching and filtering rules apply as before. You can delete a job with:

```
client.delete_job(project_id=”<project id>”,job_id=”<job_id>”)
```

Finally you can get the job ID from a job response and create a job run for a job:

```
job_run =client.create_job_run(cmlapi.CreateJobRunRequest(), project_id=”<pr
oject id>”,job_id=”<job id>”)
```

If you wish to stop the job run, you can do so as well

```
client.stop_job_run(project_id=”<project id>”,job_id=”<job id>”, run_id=job_
run.id)
```

Check the status of a job:

```
client.list_job_runs(project_id, job_id, sort="-created_at", page_size=1)
```

Using the Models API

This example demonstrates the use of the Models API. To run this example, first do the following:

1. Create a project with the Python template and a legacy engine.
2. Start a session.
3. Run !pip3 install sklearn
4. Run fit.py

The example script first obtains the project ID, then creates and deploys a model.

```
projects = client.list_projects(search_filter=json.dumps({"name": “<your pro
ject name>”}))
project = projects.projects[0] # assuming only one project is returned by
 the above query
model_body = cmlapi.CreateModelRequest(project_id=project.id, name="Demo Mo
del", description="A simple model")
model = client.create_model(model_body, project.id)
model_build_body = cmlapi.CreateModelBuildRequest(project_id=project.id, mod
el_id=model.id, file_path="predict.py", function_name="predict", kernel="pyt
hon3")
model_build = client.create_model_build(model_build_body, project.id, mod
el.id)
while model_build.status not in [“built”, “build failed”]:
 print(“waiting for model to build...”)
time.sleep(10)
 model_build = client.get_model_build(project.id, model.id, model_build.id)
if model_build.status == “build failed”:
 print(“model build failed, see UI for more information”)
 sys.exit(1)
print(“model built successfully!”)
model_deployment_body = cmlapi.CreateModelDeploymentRequest(project_id=proje
ct.id, model_id=model.id, build_id=model_build.id)
model_deployment = client.create_model_deployment(model_deployment_body,
project.id, model.id, build.id)
while model_deployment.status not in [“stopped”, “failed”, “deployed”]:
 print(“waiting for model to deploy...”)
 time.sleep(10)
 model_deployment = client.get_model_deployment(project.id, model.id, model
_build.id, model_deployment.id)
if model_deployment.status != “deployed”:
 print(“model deployment failed, see UI for more information”)
 sys.exit(1)
print(“model deployed successfully!”)
```

Using the Applications API

Here is an example of using the Application API.

```
application_request = cmlapi.CreateApplicationRequest(
     name = "application_name",
     description = "application_description",
     project_id = project_id,
     subdomain = "application-subdomain",
     kernel = "python3",
     script = "entry.py",
     environment = {"KEY": "VAL"}
)
app = client.create_application(
     project_id = project_id,
     body = application_request
```

```
)
```

Using the Cursor class

The Cursor is a helper function that works with any endpoint used for listing items, such as list_projects, list_jobs, or list_runtimes. The Cursor returns an iterable object. The following example returns a list of runtimes.

```
cursor = Cursor(client.list_runtimes)
runtimes = cursor.items()
for rt in runtimes:
    print(rt.image_identifier)
```

The Cursor can also use a search filter, as shown in this example:

```
cursor = Cursor(client.list_runtimes, search_filter = json.dumps({"image_ide
ntifier":"jupyter"}))
```

End to end example

This example creates a project, job, and job run. For the job script, it uses the analysis.py file that is included in the Python template.

```
import cmlapi
import time
import sys
import random
import string
random_id=''.join(random.choice(string.ascii_lowercase) for i in range(10))
project_body = cmlapi.CreateProjectRequest(name="APIv2 Test Project " + ran
dom_id, description="Project for testing APIv2", template="Python")
project_result = client.create_project(project_body)
poll_retries = 5
while True: # wait for the project to reach the "success" state
  project = client.get_project(project_result.id)
  if project.creation_status == "success":
    break
  poll_retries -= 1
  if poll_retries == 0:
    print("failed to wait for project creation to succeed")
    sys.exit(1)
  time.sleep(2) # wait a couple seconds before the next retry

job_body = cmlapi.CreateJobRequest(name="APIv2 Test Job " + random_id, kerne
l="python3", script="analysis.py")
job_result = client.create_job(job_body, project_id=project_result.id)
job_run = client.create_job_run(cmlapi.CreateJobRunRequest(), project_id=p
roject_result.id, job_id=job_result.id)
```

Command Line Tools in CML

Cloudera Machine Learning ships with the following command line tools. The purpose of each tool differs.

- CDP CLI for Cloudera Machine Learning - If you prefer to work in a terminal window, you can download and configure the CDP client that gives you access to the CDP CLI tool. The CDP CLI allows you to perform the

same actions as can be performed from the management console. Use this CLI to create, delete, upgrade, and manage ML workspaces on CDP.

To view all the available commands, run:

```
cdp ml help
```

To view help for a specific command, run:

```
cdp ml <operation> help
```

If you don't already have the CDP CLI set up, see *Installing the CDP CLI Client*.

- cdswctl - Cloudera Machine Learning also ships with a CLI client that you can download from the Cloudera Machine Learning web UI. This is also referred to as the Model CLI client. The cdswctl client allows you to log in, create an SSH endpoint, launch new sessions, automate model deployment, model updates, and so on.

cdswctl Command Line Interface Client

Cloudera Machine Learning ships with a CLI client that you can download from the Cloudera Machine Learning web UI. The cdswctl client allows you to perform the following tasks:

- Logging in
- Creating an SSH endpoint
- Listing sessions that are starting or running
- Starting or stopping a session
- Creating a model
- Building and deploying models
- Listing model builds and model deployments
- Checking the status of a deployment
- Redeploying a model with updated resources
- Viewing the replica logs for a model

Other actions, such as creating a project, require you to use the Cloudera Machine Learning web UI. For information about the available commands, run the following command:

```
cdswctl --help
```

Download and Configure cdswctl

This topic describes how to download the cdswctl CLI client and configure your SSH public key to authenticate CLI access to sessions.

About this task

Before you begin, ensure that the following prerequisites are met:

- You have an SSH public/private key pair for your local machine.
- You have Contributor permissions for an existing project. Alternatively, create a new project you have access to.
- If you want to configure a third-party editor, make sure the Site Administrator has not disabled remote editing for Cloudera Machine Learning.

(Optional) Generate an SSH Public/Private Key

About this task

This task is optional. If you already have an SSH public/private key pair, skip this task. The steps to create an SSH public/private key pair differ based on your operating system. The following instructions are meant to be an example and are written for macOS using ssh-keygen.

Procedure

1. Open Terminal.
2. Run the following command and complete the fields:

   ```
   ssh-keygen -t rsa -f ~/.ssh/id_rsa
   ```

   Keep the following guidelines in mind:

   - Make sure that the SSH key you generate meets the requirements for the local IDE you want to use. For example, PyCharm requires the -m PEM option because PyCharm does not support modern (RFC 4716) OpenSSH keys.
   - Provide a passphrase when you generate the key pair. Use this passphrase when prompted for the SSH key passphrase.
   - Save the SSH key to the default ~/.ssh location.

Download cdswctl and Add an SSH Key

Procedure

1. Open the Cloudera Machine Learning web UI and go to SettingsRemote Editing for your user account.
2. Download cdswctl client for your operating system.

   Unpack it, and optionally, you can add it to the *PATH* environment variable on your system.
3. Add your SSH public key to SSH public keys for session access.

   Cloudera Machine Learning uses the SSH public key to authenticate your CLI client session, including the SSH endpoint connection to the Cloudera Machine Learning deployment.

   Any SSH endpoints that are running when you add an SSH public key must also be restarted.

Initialize an SSH Endpoint

This topic describes how to establish an SSH endpoint for Cloudera Machine Learning.

About this task

Creating an SSH endpoint is also the first step to configuring a remote editor for an ML workspace.

Procedure

1. Create a configuration file at: $HOME/.cdsw/config.yaml. The contents of config.yaml should be:

   ```
   username: <username>
   url: <ML_workspace_url>
   auth:
     authtype: 1
     basic: null
     apikey: <your_api_key>
   ```

   To collect the values for these fields, first log in to your CML workspace using SSO:

   - username: The username with which you are logged into the CML workspace. Found in the top right corner of your ML workspace.

- url: The complete URL used to access the CML workspace. For example: https://ml-<randomly-generated-cluster-name>
- apikey: Go to User Settings API Keys . Copy the value of the Legacy API Key to this field.

2. Create a local SSH endpoint to Cloudera Machine Learning. Run the following command:

```
cdswctl ssh-endpoint -p <username>/<project_name> [-c <CPU_cores>] [-m <
memory_in_GB>] [-g <number_of_GPUs>] [-r <runtime ID> ]
```

The command uses the following defaults for optional parameters:

- CPU cores: 1
- Memory: 1 GB
- GPUs: 0

For example, the following command starts a session for the user milton under the customerchurn project with .5 cores, .75 GB of memory, 0 GPUs, and the Python3 kernel:

```
cdswctl ssh-endpoint -p customerchurn -c 0.5 -m 0.75
```

To create an SSH endpoint in a project owned by another user or a team, for example finance, prepend the username to the project and separate them with a forward slash:

```
cdswctl ssh-endpoint -p finance/customerchurn -c 0.5 -m 0.75
```

This command creates session in the project customerchurn that belongs to the team finance.

Information for the SSH endpoint appears in the output:

```
...
You can SSH to it using
    ssh -p <some_port> cdsw@localhost
...
```

3. Open a new command prompt and run the outputted command from the previous step:

```
ssh -p <some_port> cdsw@localhost
```

For example:

```
ssh -p 9750 cdsw@localhost
```

You will be prompted for the passphrase for the SSH key you entered in the ML workspace web UI.

Once you are connected to the endpoint, you are logged in as the cdsw user and can perform actions as though you are accessing the terminal through the web UI.

4. Test the connection.

If you run ls, the project files associated with the session you created are shown. If you run whoami, the command returns the cdsw user.

5. Leave the SSH endpoint running as long as you want to use a local IDE.

Log into cdswctl

This topic describes how to log into cdswctl.

Procedure

1. Open the Model CLI client.

2. Run the following command while specifying the actual values for the variables:

```
cdswctl login -u <workspace_url> -n <username> -y <legacy_api_key>
```

where

- *workspace_url* is the workspace URL including the protocol (http(s)://domain.com)
- *username* is your user name on the workspace
- *legacy_api_key* is the API key that you can obtain from the Cloudera Machine Learning UI. Go to Settings API Keys and copy the Legacy API Key (and not the API Key).

To see more information about the login command parameters, run

```
cdswctl login --help
```

If all goes well, then you'll see "Login succeeded".

Prepare to manage models using the model CLI

Before you can start using the model CLI to automate model deployment or to perform any other tasks, you must install the scikit-learn machine learning library for Python through the Cloudera Machine Learning web UI.

About this task

You must perform this task through the Cloudera Machine Learning web UI.

Procedure

1. Create a new project with Python through the web UI.

   Python provides sample files that you can use to create models using CLI.
2. To start a new session, go to the **Sessions** page from the left navigation panel and click new session.

   The **Start the new session** page is displayed.
3. On **Start the new session** page, select Python 3 from the Engine Kernel drop-down menu, and click Launch Session.

   A new “Untitled Session” is created.
4. From the input prompt, install the scikit-learn machine learning library for Python by running the following command:

```
!pip3 install sklearn
```

5. Open the fit.py file available within your project from the left navigation panel.

   You can use the fit.py file to create a fitted model which creates a model.pkl file that you can use to deploy the actual model.
6. Run the fit.py file by clicking Run Run all .

   The model.pkl directory is created that you can see within your project on the left navigation pane.
7. Close the session by clicking Stop.

Create a model using the CLI

This topic describes how to create models using the model CLI.

Procedure

1. Open a terminal window and log into cdswctl.

2. Obtain the project ID as described in the following steps:
   a) Run the following command:

      ```
      cdswctl projects list
      ```

      The project ID, your username, and the project name are displayed. For example:

      1: john-smith/petal-length-predictor
   b) Note the project ID, which is a number in front of your project name.

      In this case, it is "1".
3. Run the following command while specifying the project name and note the engine image ID:

   Note: The following examples are specific to projects configured to use legacy engines and projects configured to use runtimes. Be sure to use the commands appropriate to your project configuration.

   For projects configured to use legacy engines:

   ```
   cdswctl engine-images list -p <project-name>
   ```

   For example,

   ```
   cdswctl engine-images list -p john-smith/petal-length-predictor
   ```

   For projects configured to use runtimes:

   ```
   cdswctl runtimes list
   ```

   Depending on your local setup, you may get a more readable output by post-processing the result with the following command:

   ```
   cdswctl runtimes list | python3 -m json.tool
   ```

   For this example you should pick a runtime with a Python kernel and Workbench editor. Depending on your local setup, you may filter the results using the following command:

   ```
   cdswctl runtimes list | jq  '.runtimes[] | select((.editor == "Workbench")
    and (.kernel | contains("Python")))'
   ```
4. Create a model by using the following command:

   Note: The following examples are specific to projects configured to use legacy engines and projects configured to use runtimes. Be sure to use the commands appropriate to your project configuration.

   For projects configured to use legacy engines:

   ```
   cdswctl models create
   --kernel="python3"
   --targetFilePath="predict.py"
   --targetFunctionName="predict"
   --name="Petal Length Predictor"
   --cpuMillicores=1000
   --memoryMb=2000
   --description="Model of the Iris dataset"
   --replicationType=fixed
   --numReplicas=1
   --visibility="private"
   --autoBuildModel
   --autoDeployModel
   --projectId=<project ID>
   --examples='{"request":{"petal_length":1}}'
   ```

```
--engineImageId=<engine image ID from before>
```

For projects configured to use runtimes:

```
cdswctl models create
--targetFilePath="predict.py"
--targetFunctionName="predict"
--name="Petal Length Predictor"
--cpuMillicores=1000
--memoryMb=2000
--description="Model of the Iris dataset"
--replicationType=fixed
--numReplicas=1
--visibility="private"
--autoBuildModel
--autoDeployModel
--projectId=<project ID>
--examples='{"request":{"petal_length":1}}'
--runtimeId=<runtime ID obtained above>
```

If the command runs successfully, the system displays the model details in a JSON format.

5. For more information about the models create command parameters, run the following command:

```
cdswctl models create --help
```

Build and deployment commands for models

Models have separate parameters for builds and deployments. When a model is built, an image is created. Whereas, the deployment is the actual instance of the model. You can list model builds and deployment, and monitor their state using from model CLI client (cdswctl).

Listing a model

To list the models, run the following command:

```
cdswctl models list
```

Monitoring the status of the model

To monitor the status of the build for a particular model, use the following command:

```
cdswctl models listBuild --modelId <model_ID> --projectId <project_ID>
```

You can use the --latestModelDeployment flag to get the build for the latest deployment.

Listing a deployment

To list the deployment for a particular model, run the following command:

```
cdswctl models listDeployments --modelId <model_ID>
```

Checking the status of a deployment

To check the status of your deployment, run the following command:

```
cdswctl models listDeployments --statusSet=deployed
```

Following is a list of arguments for the statusSet parameter:

- deployed
- deploying
- failed
- pending
- stopping
- stopped

Note: You can use the parameter more than once in a command to check multiple statuses of your deployed models. For example,

```
cdswctl models listDeployments --statusSet=deployed --statusSet=stop
ped --statusSet=failed
```

Deploy a new model with updated resources

You can republish a previously-deployed model in a new serving environment with an updated number of replicas or memory/CPU/GPU allocation by providing the model build ID of the model you want to rebuild.

To deploy a new model, use the following command:

```
cdswctl models deploy --modelBuildId=<build_ID> --cpuMilli
cores=<num_of_cpu_cores> --memoryMb=<memory_in_mb> --numRepli
cas=<num_of_replicas> --replicationType=<replication_type>
```

For example:

```
cdswctl models deploy --modelBuildId=<build_ID> --cpuMillicores=1200 --mem
oryMb=2200 --numReplicas=2 --replicationType=fixed
```

Note: You must specify values for all the non-zero resources, even if you do not wish to update their values. For example, in your existing deployment, if you set the cpuMillicores capacity to 1200 and you do not wish to increase or decrease it, you must still specify cpuMillicores=1200 in the command.

View replica logs for a model

When a model is deployed, Cloudera Machine Learning enables you to specify the number of replicas that must be deployed to serve requests. If a replica crashes or fails to come up, you can diagnose it by viewing the logs for every replica using the model CLI.

Procedure

1. Obtain the modelReplicaId by using the following command:

   ```
   cdswctl models listReplicas --modelDeploymentId=<model_deployment_ID>
   ```

   where the *model_deployment_ID* is the ID of a successfully deployed model.
2. To view the replica logs, run the following command:

   ```
   cdswctl models getReplicaLogs --modelDeploymentId=<model_deployment_ID> --
   modelReplicaId="<replica_ID>" --streams=stdout
   ```

   For example:

   ```
   cdswctl models getReplicaLogs --modelDeploymentId=2 --modelReplicaId="pe
   tal-length-predictor-1-2-6d6496b467-hp6tz" --streams=stdout
   ```

   The valid values for the streams parameter are stdout and stderr.

cdswctl command reference

You can manage your Cloudera Machine Learning Workbench cluster with the CLI client (cdswctl) that exists within the Cloudera Machine Learning Workbench. Running `cdswctl` without any arguments prints a brief description of each command.

Table 35: Model CLI Command Reference

Command	Description and usage
cdswctl login	Enables you to log into the model CLI client
cdswctl projects list	Lists the projects
cdswctl models create	Creates a model with the specified parameters
cdswctl models list	Lists all models You can refine the search by specifying the modelId
cdswctl models listBuild	Llists the builds for a model You can monitor the status of the build by specifying the modelId and the projectId
cdswctl models listDeployments	List the deployments for a model You can refine the search by specifying the modelId Use the statusSet parameter to check the status of the model being deployed
cdswctl models deploy	Deploys a model with the specified parameters
cdswctl models listReplicas	Enables you to view the list of model replicas You also need this information to obtain replica logs
cdswctl models getReplicaLogs	Enables you to view the logs for a model replica
cdswctl models restart	Restarts a model Usage: cdswctl models restart --modelDeploymentId=<*deployment_ID*> Note: Running this command does not change the resources if you previously ran the cdswctl models update command
`cdswctl models update`	Changes the name, description, or visibility of the model To change a model's resources, use the `cdswctl models deploy` command
`cdswctl models delete`	Deletes a model Usage: cdswctl models delete --id=<*model_ID*>

Data Access

Cloudera Machine Learning is a flexible, open platform supporting connections to many data sources.

CML supports easy, secure data access through connection snippets and the cml.data library. This library, implemented in Python, abstracts all of the complexity of configuring, initializing, and authenticating data

connections. Users choosing to manually create and configure the data connections can follow the reference guide below.

Upload and work with local files

This topic includes code samples that demonstrate how to access local data for CML workloads.

If you want to work with existing data files (.csv, .txt, etc.) from your computer, you can upload these files directly to your project in the CML workspace. Go to the project's Overview page. Under the Files section, click Upload and select the relevant data files to be uploaded. These files will be uploaded to an NFS share available to each project.

Note: Storing large data files in your Project folder is highly discouraged. You can store your data files in the Data Lake.

The following sections use the tips.csv dataset to demonstrate how to work with local data stored in your project. Before you run these examples, create a folder called data in your project and upload the dataset file to it.

Python

```
import pandas as pd
tips = pd.read_csv('data/tips.csv')

tips \
  .query('sex == "Female"') \
  .groupby('day') \
  .agg({'tip' : 'mean'}) \
  .rename(columns={'tip': 'avg_tip_dinner'}) \
  .sort_values('avg_tip_dinner', ascending=False)
```

R

```
library(readr)
library(dplyr)

# load data from .csv file in project
tips <- read_csv("data/tips.csv")

# query using dplyr
tips %>%
  filter(sex == "Female") %>%
  group_by(day) %>%
  summarise(
    avg_tip = mean(tip, na.rm = TRUE)
  ) %>%
  arrange(desc(avg_tip))
```

Connect to CDW

The Data Connection Snippet feature now suggests using the cml.data library to connect to CDW virtual warehouses - these code snippets pop up as suggestions for every new session in a project. For further information, see *Using data connection snippets*.

However, if you would still like to use raw Python code to connect, follow the below details.

You can access data stored in the data lake using a Cloudera Data Warehouse cluster from a CML workspace, using the impyla Python package.

Configuring the connection

The CDW connection requires a WORKLOAD_PASSWORD that can be configured following the steps described in *Setting the workload password*, linked below.

The VIRTUAL_WAREHOUSE_HOSTNAME can be extracted from the JDBC URL that can be found in CDW, by selecting the Option menu > Copy JDBC String on a Virtual Warehouse.

For example, if the JDBC string copied as described above is:

```
jdbc:impala//<your-vw-host-name.site>/default;transportMode=http;httpPath=cl
iservice;socketTimeout=60;ssl=true;auth=browser;
```

Then, the extracted hostname to assign to the VWH_HOST is: <your-vw-host-name.site>

Connection code

Enter this code in your project file, and run it in a session.

```
# This code assumes the impyla package to be installed.
# If not, please pip install impyla

from impala.dbapi import connect
import os
USERNAME=os.getenv(HADOOP_USER_NAME)
PASSWORD=os.getenv(WORKLOAD_PASSWORD)
VWH_HOST = "<<VIRTUAL_WAREHOUSE_HOSTNAME>>"
VWH_PORT = 443
conn = connect(host=VWH_HOST, port=VWH_PORT, auth_mechanism="LDAP", user=USE
RNAME,  password=PASSWORD, use_http_transport=True, http_path="cliservice",
use_ssl=True)

dbcursor = conn.cursor()
dbcursor.execute("<<INSERT SQL QUERY HERE>>")
for row in dbcursor:
  print(row)

#Sample pandas code
#from impala.util import as_pandas
#import pandas
#dbcursor = conn.cursor()
#dbcursor.execute("<<INSERT SQL QUERY HERE>>")
#tables = as_pandas(cursor)
#tables
#dbcursor.close()
```

Accessing data with Spark

When you are using CDW, you can use JDBC connections.

JDBC is useful in the following cases:

1. Use JDBC connections when you have fine-grained access.
2. If the scale of data sent over the wire is on the order of tens of thousands of rows of data.

Add the Python code as described below, in the Session where you want to utilize the data, and update the code with the data location information.

Permissions

In addition, check with the Administrator that you have the correct permissions to access the data lake. You will need a role that has read access only.

How to obtain the Data Lake directory location

You need this location if you are using a Direct Reader connection.

1. In the CDP home page, select Management Console.
2. In Environments, select the environment you are using.
3. In the tabbed section, select Cloud Storage.
4. Choose the location where your data is stored.
5. For managed data tables, copy the location shown for Hive Metastore Warehouse.
6. For external unmanaged data tables, copy the location shown for Hive Metastore External Warehouse.
7. Paste the location into the connection script in the designated position. If you are using AWS, the location starts with s3:, and if you are using Azure, it starts with abfs:. If you are using a different location in the data lake, the default path is shown by Hbase Root.

Set up a JDBC Connection

When using a JDBC connection, you read through a virtual warehouse that has Hive or Impala installed. You need to obtain the JDBC connection string, and paste it into the script in your session.

1. In CDW, go to the Hive database containing your data.
2. From the kebab menu, click Copy JDBC URL.
3. Paste it into the script in your session.
4. You also have to enter your user name and password in the script. You should set up environmental variables to store these values, instead of hardcoding them in the script.

Use JDBC Connection with PySpark

PySpark can be used with JDBC connections, but it is not recommended. The recommended approach is to use Impyla for JDBC connections. For more information, see *Connect to CDW*.

Procedure

1. In your session, open the workbench and add the following code.
2. Obtain the JDBC connection string, as described above, and paste it into the script where the “jdbc” string is shown. You will also need to insert your user name and password, or create environment variables for holding those values.

Example

This example shows how to read external Hive tables using Spark and a Hive Virtual Warehouse.

```
from pyspark.sql import SparkSession
from pyspark_llap.sql.session import HiveWarehouseSession

spark = SparkSession\
.builder\
.appName("CDW-CML-JDBC-Integration")\
.config("spark.security.credentials.hiveserver2.enabled","false")\
.config("spark.datasource.hive.warehouse.read.jdbc.mode", "client")\
.config("spark.sql.hive.hiveserver2.jdbc.url",
"jdbc:hive2://hs2-aws-2-hive-viz.env-j2ln9x.dw.ylcu-atmi.cloudera.site/def
ault;\
transportMode=http;httpPath=cliservice;ssl=true;retries=3;\
user=<username>;password=<password>")\
.getOrCreate()

hive = HiveWarehouseSession.session(spark).build()
hive.showDatabases().show()
hive.setDatabase("default")
hive.showTables().show()
hive.sql("select * from foo").show()
```

```
```

Related Information

Connect to CDW

Connect to a CDP Data Hub cluster

The Data Connection Snippet feature helps simplify the user experience by abstracting the complexity of creating and configuring a data connection.

You can set up a Data Hub Impala or Hive connection by following the documentation: *Set up a data connection to CDP Data Hub*.

However, if you would still like to use raw Python code to connect, follow this Python example:

```
from impala.dbapi import connect

#Example connection string:
# jdbc:hive2://my-test-master0.eng-ml-i.svbr-nqvp.int.cldr.work/;ssl=true;
transportMode=http;httpPath=my-test/cdp-proxy-api/hive

USERNAME=os.getenv(HADOOP_USER_NAME)
PASSWORD=os.getenv(WORKLOAD_PASSWORD)

conn = connect(
    host = "my-test-master0.eng-ml-i.svbr-nqvp.int.cldr.work",
    port = 443,
    auth_mechanism = "LDAP",
    use_ssl = True,
    use_http_transport = True,
    http_path = "my-test/cdp-proxy-api/hive",
    user = USERNAME,
    password = PASSWORD)
cursor = conn.cursor()
cursor.execute("<<INSERT SQL QUERY HERE>>")

for row in cursor:
    print(row)
cursor.close()
conn.close()
```

Connect to external Amazon S3 buckets

Every language in Cloudera Machine Learning has libraries available for uploading to and downloading from Amazon S3.

To work with external S3 buckets in Python, do the following:

- Add your Amazon Web Services access keys to your project's environment variables as AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.

Python

```
# Install Boto to the project
%pip install boto3

import boto3

s3 = boto3.client('s3')
# Print out bucket names
```

```
for bucket in s3.buckets.all():
  print(bucket.name)

# Download a file
s3.download_file('BUCKET_NAME', 'OBJECT_NAME', 'FILE_NAME')
```

Connect to External SQL Databases

Every language in Cloudera Machine Learning has multiple client libraries available for SQL databases.

If your database is behind a firewall or on a secure server, you can connect to it by creating an SSH tunnel to the server, then connecting to the database on localhost.

If the database is password-protected, consider storing the password in an environmental variable to avoid displaying it in your code or in consoles. The examples below show how to retrieve the password from an environment variable and use it to connect.

Python

You can access data using pyodbc or SQLAlchemy

```
# pyodbc lets you make direct SQL queries.
!wget https://pyodbc.googlecode.com/files/pyodbc-3.0.7.zip
!unzip pyodbc-3.0.7.zip
!cd pyodbc-3.0.7;python setup.py install --prefix /home/cdsw
import os

# See http://www.connectionstrings.com/ for information on how to construct
 ODBC connection strings.
db = pyodbc.connect("DRIVER={PostgreSQL Unicode};SERVER=localhost;PORT=54
32;DATABASE=test_db;USER=cdswuser;OPTION=3;PASSWORD=%s" % os.environ["POSTGR
ESQL_PASSWORD"])
cursor = cnxn.cursor()
cursor.execute("select user_id, user_name from users")

# sqlalchemy is an object relational database client that lets you make data
base queries in a more Pythonic way.
!pip install sqlalchemy
import os

import sqlalchemy
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
db = create_engine("postgresql://cdswuser:%s@localhost:5432/test_db" % os.en
viron["POSTGRESQL_PASSWORD"])
session = sessionmaker(bind=db)
user = session.query(User).filter_by(name='ed').first()
```

R

You can access remote databases with dplyr.

```
install.packages("dplyr")
library("dplyr")
db <- src_postgres(dbname="test_db", host="localhost", port=5432, user="cds
wuser", password=Sys.getenv("POSTGRESQL_PASSWORD"))
flights_table <- tbl(db, "flights")
select(flights_table, year:day, dep_delay, arr_delay)
```

Accessing Ozone from Spark

In CML, you can connect Spark to the Ozone object store with a script. The following example demonstrates how to do this.

This script, in Scala, counts the number of word occurrences in a text file. The key point in this example is to use the following string to refer to the text file: o3fs://hivetest.s3v.o3service1/spark/jedi_wisdom.txt

Word counting example in Scala

```
import sys.process._

// Put the input file into Ozone
//"hdfs dfs -put data/jedi_wisdom.txt o3fs://hivetest.s3v.o3service1/spark
" !
// Set the following spark setting in the file "spark-defaults.conf" on the
 CML session using terminal
//spark.yarn.access.hadoopFileSystems=o3fs://hivetest.s3v.o3service1.neptune
e01.olympus.cloudera.com:9862

//count lower bound
val threshold = 2
// this file must already exist in hdfs, add a
// local version by dropping into the terminal.
val tokenized = sc.textFile("o3fs://hivetest.s3v.o3service1/spark/jedi_wisdo
m.txt").flatMap(_.split(" "))
// count the occurrence of each word
val wordCounts = tokenized.map((_ , 1)).reduceByKey(_ + _)
// filter out words with fewer than threshold occurrences
val filtered = wordCounts.filter(_._2 >= threshold)
System.out.println(filtered.collect().mkString(","))
```

Built-in CML Visualizations

You can use built-in CML tools to create data visualizations including simple plots, saved images, HTML and iFrame visualizations, and grid displays.

Simple Plots

Cloudera Machine Learning supports using simple plot to create data visualizations.

To create a simple plot, run a console in your favorite language and paste in the following code sample:

R

```
# A standard R plot
plot(rnorm(1000))
# A ggplot2 plot
library("ggplot2")
qplot(hp, mpg, data=mtcars, color=am,
facets=gear~cyl, size=I(3),
xlab="Horsepower", ylab="Miles per Gallon")
```

Python

```
import matplotlib.pyplot as plt
import random
plt.plot([random.normalvariate(0,1) for i in xrange(1,1000)])
```

Cloudera Machine Learning processes each line of code individually (unlike notebooks that process code per-cell). This means if your plot requires multiple commands, you will see incomplete plots in the workbench as each line is processed.

To get around this behavior, wrap all your plotting commands in one Python function. Cloudera Machine Learning will then process the function as a whole, and not as individual lines. You should then see your plots as expected.

Saved Images

You can display images within your reports.

Use the following commands:

R

```
library("cdsw")

download.file("https://upload.wikimedia.org/wikipedia/commons/2/29/Minard.
png", "/cdn/Minard.png")
image("Minard.png")
```

Python

```
import urllib
from IPython.display import Image
urllib.urlretrieve("http://upload.wikimedia.org/wikipedia/commons/2/29/Minar
d.png", "Minard.png")

Image(filename="Minard.png")
```

HTML Visualizations

Your code can generate and display HTML in Cloudera Machine Learning.

To create an HTML widget, paste in the following:

R

```
library("cdsw")
html('<svg><circle cx="50" cy="50" r="50" fill="red" /></svg>')
```

Python

```
from IPython.display import HTML
HTML('<svg><circle cx="50" cy="50" r="50" fill="red" /></svg>')
```

Scala

Cloudera Machine Learning allows you to build visualization libraries for Scala using jvm-repr. The following example demonstrates how to register a custom HTML representation with the "text/html" mimetype in Cloudera Machine Learning. This output will render as HTML in your workbench session.

```
//HTML representation
case class HTML(html: String)
//Register a displayer to render html
Displayers.register(classOf[HTML],
  new Displayer[HTML] {
    override def display(html: HTML): java.util.Map[String, String] = {
      Map(
        "text/html" -> html.html
      ).asJava
    }
  })
```

```
val helloHTML = HTML("<h1> <em> Hello World </em> </h1>")

display(helloHTML)
```

IFrame Visualizations

Most visualizations require more than basic HTML. Embedding HTML directly in your console also risks conflicts between different parts of your code. The most flexible way to embed a web resource is using an IFrame.

Note:

Cloudera Machine Learning versions 1.4.2 (and higher) added a new feature that allowed users to HTTP security headers for responses to Cloudera Machine Learning. This setting is enabled by default. However, the X-Frame-Options header added as part of this feature blocks rendering of iFrames injected by third-party data visualization libraries.

To work around this issue, a site administrator can go to the Admin Security page and disable the Enable HTTP security headers property. Restart Cloudera Machine Learning for this change to take effect.

R

```
library("cdsw")
iframe(src="https://www.youtube.com/embed/8pHzROP1D-w", width="854px", heigh
t="510px")
```

Python

```
from IPython.display import HTML
HTML('<iframe width="854" height="510" src="https://www.youtube.com/embed/8
pHzROP1D-w"></iframe>')
```

You can generate HTML files within your console and display them in IFrames using the /cdn folder. The cdn folder persists and services static assets generated by your engine runs. For instance, you can embed a full HTML file with IFrames.

R

```
library("cdsw")
f <- file("/cdn/index.html")
html.content <- paste("<p>Here is a normal random variate:", rnorm(1), "</
p>")
writeLines(c(html.content), f)
close(f)
iframe("index.html")
```

Python

```
from IPython.display import HTML
import random

html_content  = "<p>Here is a normal random variate: %f </p>" % random.norma
lvariate(0,1)

file("/cdn/index.html", "w").write(html_content)
HTML("<iframe src=index.html>")
```

Cloudera Machine Learning uses this feature to support many rich plotting libraries such as htmlwidgets, Bokeh, and Plotly.

Grid Displays

Cloudera Machine Learning supports built-in grid displays of DataFrames across several languages.

Python

Using DataFrames with the pandas package requires per-session activation:

```
import pandas as pd
pd.DataFrame(data=[range(1,100)])
```

For PySpark DataFrames, use pandas and run df.toPandas() on a PySpark DataFrame. This will bring the DataFrame into local memory as a pandas DataFrame.

Note:

A Python project originally created with engine 1 will be running pandas version 0.19, and will not auto-upgrade to version 0.20 by simply selecting engine 2 in the project's Settings Engine page.

The pandas data grid setting only exists starting in version 0.20.1. To upgrade, manually install version 0.20.1 at the session prompt.

```
!pip install pandas==0.20.1
```

R

In R, DataFrames will display as grids by default. For example, to view the Iris data set, you would just use:

```
iris
```

Similar to PySpark, bringing Sparklyr data into local memory with as.data.frame will output a grid display.

```
sparkly_df %>% as.data.frame
```

Scala

Calling the display() function on an existing dataframe will trigger a collect, much like df.show().

```
val df = sc.parallelize(1 to 100).toDF()
display(df)
```

Documenting Your Analysis

Cloudera Machine Learning supports Markdown documentation of your code written in comments.

This allows you to generate reports directly from valid Python and R code that runs anywhere, even outside Cloudera Machine Learning. To add documentation to your analysis, create comments in Markdown format:

R

```
# Heading
# -------
#
# This documentation is **important.**
#
# Inline math: $e^ x$
#
# Display math: $$y = \Sigma x + \epsilon$$

print("Now the code!")
```

Python

```
# Heading
# -------
#
# This documentation is **important.**
#
# Inline math: $e^ x$
#
# Display math: $$y = \Sigma x + \epsilon$$

print("Now the code!")
```

Cloudera Data Visualization for ML

Cloudera Data Visualization enables you to explore data and communicate insights across the whole data lifecycle by using visual objects. The fast and easy self-service data visualization streamlines collaboration in data analytics through the common language of visuals.

Using this rich visualization layer enables you to accelerate advanced data analysis. The web-based, no-code, drag-and-drop user interface is highly intuitive and enables you to build customized visualizations on top of your datasets, build dashboards and applications, and publish them anywhere across the data lifecycle. This solution allows for customization and collaboration, and it provides you with a dynamic and data-driven insight into your business.

Cloudera Data Visualization is integrated with Cloudera Machine Learning (CML) in all form factors. You can use the same visualization tool for structured, unstructured/text, and ML analytics, which means deeper insights and more advanced dashboard applications. You can create native data visualizations to provide easy predictive insights for business users and accelerate production ML workflows from raw data to business impact.

For more information, see the *Cloudera Data Visualization documentation*.

Related Information

Cloudera Data Visualization in CDP Public Cloud

Cloudera Data Visualization in CDSW

Jupyter Magic Commands

Cloudera Machine Learning's Scala and Python kernels are based on Jupyter kernels. Jupyter kernels support varying magic commands that extend the core language with useful shortcuts. This section details the magic commands (magics) supported by Cloudera Machine Learning.

Line magics begin with a single %: for example, %timeit. Cell magics begin with a double %%: for example, %%bash.

Python

In the default Python engine, Cloudera Machine Learning supports most line magics, but no cell magics.

Cloudera Machine Learning supports the shell magic !: for example, !ls -alh /home/cdsw.

Cloudera Machine Learning supports the help magics ? and ??: for example, ?numpy and ??numpy. ? displays the docstring for its argument. ?? attempts to print the source code. You can get help on magics using the ? prefix: for example, ?%timeit.

Cloudera Machine Learning supports the line magics listed at https://ipython.org/ipython-doc/3/interactive/magics.html#line-magics, with the following exceptions:

- %colors
- %debug

- %edit
- %gui
- %history
- %install_default_config
- %install_profiles
- %lsmagic
- %macro
- %matplotlib
- %notebook
- %page
- %pastebin
- %pdb
- %prun
- %pylab
- %recall
- %rerun
- %save
- %sc

Related reference

Scala

Scala

Cloudera Machine Learning's Scala kernel is based on Apache Toree. It supports the line magics documented in the Apache Toree magic tutorial.

Related reference

Python

Release Notes

Find out about the latest features of each release here. Also, descriptions and workarounds for some known issues are described here.

What's New

Major features and updates for the Cloudera Machine Learning service on Private Cloud.

November 18, 2022

CML on Private Cloud, version 1.4.1, has the following updates.

New features and updates

- Legacy CDSW Cluster Detected - After the upgrade to Private Cloud 1.4.1, if the base cluster contains a CDSW installation, you will see a message recommending you to upgrade the cluster. Do NOT click Upgrade as this feature is not yet GA.
- HDFS transparent encryption - Encryption at rest (HDFS transparent encryption) is supported in CML.
- TSB-628 - The issue in this Technical Service Bulletin, "Sensitive user data getting collected in CML/CDSW workspace diagnostic bundles" was resolved.

June 22, 2022

CML on Private Cloud, version 1.4.0, has the following updates.

New features and updates

- Model metrics visualization - This feature allows Data Scientists and Machine Learning Engineers to monitor technical metrics relating to their running models, such as resource consumption and request throughput, within Cloudera Machine Learning.

Fixed issues

- DSE-19937 - Fixed an issue where the pagination widget on the Session list page may not function as expected.
- DSE-20085 - Fixed a bug where Job report recipients who subscribed to notification emails when their jobs terminated, may receive notification emails for termination statuses that they did not subscribe to.
- DSE-19751Fixed a bug where projects may not be sorted correctly on the project list page when using the Created By field for sorting.

April 11, 2022

CML on Private Cloud, version 1.3.4, has the following updates.

New features and updates

- ML Runtimes - ML Runtimes are now supported in CML Private Cloud. For more information, see Managing ML Runtimes.
- Cloudera Data Visualization - Cloudera Data Visualization is now available in the default runtime.
- GPU Taint support - GPU taints, which affect node scheduling, are now supported for both OCP and ECS clusters. For more information, see GPU node setup.

January 13, 2022

CML on Private Cloud, version 1.3.3, has the following updates.

New features and updates

- Business User Experience - A new user role, ML BusinessUser, provides restricted access to view Applications created in CML.
- API v2 - A new API for operations on projects, jobs, models, and applications is now generally available.

Installation notes

- Upgrade - It is not possible to upgrade an existing ML workspace on an ECS cluster. You have to provision a new workspace.
- Upgrade - After upgrading the ECS control plane, model and experiment building in an ML workspace might fail. See the Known Issue for more information.
- Engines - Engine version 15-cml-2021.09-2 has been patched for CVE-2021-44228, the Apache Log4j2 vulnerability. You should use engine version 15-cml-2021.09-2 instead of version 15-cml-2021.09-1 wherever possible.

October 29, 2021

CML on Private Cloud, version 1.3.2, has the following updates.

Installation notes

- Upgrade - Fixed an issue so that upgrading a CML workspace with ML Governance enabled works.

October 4, 2021

CML on Private Cloud, version 1.3.1, has the following new features and updates.

New features and updates

- Embedded Container Service (ECS) is now supported.

Installation notes

- Installation - If ECS is installed using Cloudera Docker Registries, then CML Workspace Model and Experiment building is not supported.
- Upgrade - Upgrading a CML workspace with ML Governance enabled fails.

April 27, 2021

CML on Private Cloud, version 1.2, has the following new features and updates.

New features and updates

- Support for OCP 4.6 and upgrading from PVC 1.1.
- Improved non-transparent proxy support for air-gapped environments.
- Introduced Applied ML Prototypes (AMPs).
- Added NFS support:
  - NFS versions v3 and v4.x are supported.
  - External NFS security improvements - no_root_squash export option has been removed.
- Support added for custom service principals (Beta).
- Monitoring now uses CDP centralized Grafana. Added database metrics and improved alerts.

Bug fixes

- DSE-12037 - Fixed an issue with the seamless login for Grafana.
- DSE-14891 - Fixed an issue with broken Engine and Session log links.
- Various security fixes.

December 16, 2020

CML on Private Cloud, version 1.1, has the following new features and updates.

- MLOPS-216 - Production ML Support

  Model Metrics track machine model serving performance metrics. Model Governance use Apache Atlas to track builds, experiments and deployment of machine learning models.
- DSE-10777 - UMS Integration

  MLUser and MLAdmin resource roles are now available and assignable through Environment settings.
- DSE-12955 - Self Signed Private CA certs For custom container registries

  Customers can now use Container registries that are using self signed or private CA signed certificates. There is an option to upload the self signed or private CA signed certificates certificate during Private Cloud installation.
- DSE-10759 - GPU support

  The OpenShift Nvidia operator is now supported for use with CML workloads.

August 17, 2020

This is the first release of CML on Private Cloud, version 1.0.

CML on Private Cloud lets you:

- Run Machine Learning workloads on OpenShift clusters in your own data center.
- Easily onboard a new tenant and provision an ML workspace in a shared OpenShift environment.
- Enable data scientists to access shared data on CDP Private Cloud Base and CDW.
- Leverage Spark-on-K8s to spin up and down Spark clusters on demand.
- Take advantage of most CML features on public cloud, including Teams, Projects, Experiments, Models, and Applications.

Related Information

Known Issues and Limitations

Known Issues and Limitations

You might run into some known issues while using Cloudera Machine Learning on Private Cloud.

DSE-23329: Spark API fails when reading or writing to Ozone filesystem

CML with Spark as a runtime addon does not work with the Ozone file system.

DSE-33636: Workloads unable to start up after changing default hadoopCLI addon

Changing the default Hadoop CLI Runtime Addon causes jobs, models, and application workloads to be unable to start up.

Workarounds:

1. Open affected workload settings.
2. Update the workload (this updates the Hadoop CLI Addon associated with the workload to the default one.)
3. For Jobs: update.
4. For Applications: update and restart.
5. For Models: deploy a new build.

Please see Machine Learning Site administration>Disable Addons for related tasks.

DSE-20923: User search requires lower case

When searching for a user name in Site Administration Users , you must enter only lower-case letters for the name you are searching for. Lower-case letters will match upper-case letters in the target name.

DSE-19036: Model and experiment building fails after ECS upgrade

After ECS Control Plane upgrade, your ML workspace may fail to perform Model and Experiment building with a message like: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

To resolve this problem, perform these steps:

1. Access Cloudera Manager
2. Navigate to the Embedded Cluster ECS Web UI: Clusters Your Embedded Cluster ECS Web UI ECS Web UI
3. Select the namespace of your ML workspace on the top left dropdown.
4. In Workloads Deployments , locate s2i-builder in the list.
5. In the Action menu for s2i-builder, select Restart.

DSE-13117: Container Image Registries assuming mutual TLS for authentication are not supported

If Private Cloud images are hosted in an image registry assuming mutual TLS for authentication, this will cause Model deployments and Experiments to fail. Mutual TLS registries are not supported.

DSE-12541: Self Signed Certificates for Container Registry cause Models and Experiments to fail

If you are using self-signed or Private CA signed certificates for Container image registry authentication, model deployements and experiments will fail with an error similar to: Error initializing source docker://<registryIP>:5000/alpine:latest: error pinging docker registry <registryIP>:5000: Get https://<registryIP>:5000/v2/: x509: certificate signed by unknown authority

As a workaround, create a ConfigMap in the namespace where the CML workspace is installed.

1. Create a ConfigMap as shown in this example. Here, <namespace> indicates the workspace where the CML workspace is installed.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: <external-registry-name>
  Namespace: <namespace>
data:
  registry.crt: |
    -----BEGIN CERTIFICATE-----
    < certificate content goes here >
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    < certificate content goes here >
    -----END CERTIFICATE-----
```

2. Mount the ConfigMap to the s2i-builder deployment as shown here. Add the following mountPath in the volu meMounts section for the s2i-builder pod:

```
- mountPath: /etc/docker/certs.d/<registry>[:portnum]
  name: external-registry
```

   Under the volumes section, add the ConfigMap reference:

```
- configMap:
     defaultMode: 420
     name: <external-registry-name>
  name: external-registry
```

3. Run the following command and check the output. Note in particular the mountPath and configMap specifications at the end.

```
# kubectl get deployment s2i-builder -n <namespace> -o yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  [...]
  name: s2i-builder
  namespace: <namespace/workspacename>
  [...]
spec:
  [...]
  template:
    [...]
    spec:
    [...]
    containers:
    - name: s2i-builder
    [...]
    volumeMounts:
          [...]
          - mountPath:/etc/docker/certs.d/<registry>[:portnum]
            name: external-registry
      [...]
  volumes:
  [...]
  - configMap:
      defaultMode: 420
      name: <external-registry-name>
    name: external-registry
status:
```

```
[...]
```

DSE-12367: s2i-queue pod goes into CrashLoop Failure causing ML workspace installation to fail

CML workspace install can fail because the s2i-queue pod may be stuck in a CrashLoop Failure. The error in the logs might look similar to: Failed to create thread: Resource temporarily unavailable (11) /usr/lib/rabbitmq/bin/rabbitmq-server: line 182: 45 Aborted (core dumped) start_rabbitmq_server "$@" Only root or rabbitmq can run rabbitmq-server

To fix this, apply the following workaround: # kubectl set env statefulset/s2i-queue RABBITMQ_IO_THREAD_POOL_SIZE="50" -n <namespace>

DSE-12329: Email invitation feature

The feature to invite new users by email does not work in Public or Private cloud, but it still appears in the UI.

DSE-12289: Airgap support: Proxies are not supported in CML Private Cloud 1.0

Use of a proxy server, for example for external internet connectivity for an airgap cluster, is not supported. Transparent proxies, however, should work normally.

DSE-12238: Create Project request takes longer than timeout

If a Create Project request takes longer than a certain timeout, a second request might be submitted. If this happens, multiple projects with similar names might be created.

As a workaround, create an empty project, create a session inside the project, then git clone your project inside a workbench terminal. Additionally, you can upload a zip file or a folder using the file preview table.

If multiple forks are created, delete the extra ones.

DSE-12090: User displays as unknown in Event History

In the Event History on the workspace Events tab, a user may display as unknown if they are authenticated by LDAP.

Fix: The user needs to be assigned the IamViewer role to view these details.

DSE-11979: Certificate failure when pulling images from the S2I container registry

During Model or Experiment deployment, a certificate failure similar Failed to pull image x509: certificate signed by unknown authority can occur. This is due to a Red Hat issue with OpenShift Container Platform 4.3.x where the image registry cluster operator configuration must be set to Managed.

To set the configuration, first apply a patch using this command: # oc patch configs.imageregistry.operator.openshift.io cluster --patch '{"spec":{"managementState":"Managed"}}'

Next, run the following command: # oc get config cluster -o yaml

The managementState is now set to Managed.

DSE-11870: Hung File, Stale File, and Fork issues with NFS

Hung File Operations: Certain file operations, such as `stat(2)` or `stat(1)` might stop responding, and if the file operation was performed through the CML web UI, the web operation might timeout. This indicates an NFS server that is not reachable for some reason. The error might manifest itself on the web UI when you try to open an ML project as an HTTP error, code 500. Check the logs for error messages similar to the following:

```
 2020-07-13 22:42:23.914 1 ERROR AppServer.Lib.Utils Finish grpc, failed data =
        [{"rpc":"1","service":"2","reqId":"3","err":"4"},"stat","VFS","18a07980-c55a-11ea-9bb9-a35829b422d9",{"message":"
```

```
        5","stack":"6","code":4,"metadata":"7","details":"8","futureStack"
:"6"},"4
        DEADLINE_EXCEEDED: Deadline Exceeded","Error: 4 DEADLINE_EXCEEDED:
Deadline Exceeded\n at
        Object.exports.createStatusError (/home/cdswint/services
        /web/node_modules/grpc/src/common.js:91:15)\n at Object.onReceiveSta
tus
        (/home/cdswint/services/web/node_modules/grpc/src/client_interceptor
s.js:1209:28)\n at
        InterceptingListener._callNext (/home/cdswint/services/web/
        node_modules/grpc/src/client_interceptors.js:568:42)\n at
        InterceptingListener.onReceiveStatus
        (/home/cdswint/services/web/node_modules/grpc/src/client_intercept
ors.js:618:8)\n at
        callback (/home/cdswint/services/web/n
        ode_modules/grpc/src/client_interceptors.js:847:24)",{"_internal_r
epr":"9","flags":0},"Deadline
        Exceeded",{}]
```

Solution: Check your NFS server and make sure it is running. You will need to restart the NFS clients in your ML workspace's namespace. These are the "ds-vfs" and "s2i-client" pods. Simply delete the Kubernetes pods whose names start with "ds-vfs" and "s2i-client".

Stale File Handles: When opening a project from the ML web UI, an error message like "NFS: Stale file handle" shows up on the UI.

Solution: This is indicative of an NFS server and a client being out of sync, probably caused by a server restart along with file system content change on the server that the client is not aware of. You should restart NFS client pods in your ML workspace's namespace. The are the "ds-vfs", "s2i-client", and any user sessions that are affected by the "Stale file handle" error.

Project Fork Creating Multiple Copies: When creating a new project from an existing project using the "Fork" feature, you might see the operation seemingly fail on the UI, but it still ends up creating multiple copies of the source project.

Solution: This issue happens when forking a project takes longer than the idle connection timeout set on the external load balancer, as well as in HA Proxy policy settings on OpenShift. Increase the idle connection timeout to at least 5 minutes. Depending on the performance of the NFS server, a higher timeout may be necessary.

DSE-11837: Timeout limitation for Project API

If you create a project in the UI using git clone, you may get the error message Whoops, there was an unexpected error. If you create a project using the API, a timeout may occur.

Prerequisites for CML in Private Cloud:

- Set any external load balancer server timeout to 5 min.

For a TLS Enabled Workspace:

- Set the annotation haproxy.router.openshift.io/timeout=300 on each route in a deployed CML workspace namespace:

  ```
  oc annotate route --all=true --overwrite=true -n
                <cml-namespace> haproxy.router.openshift.io/timeout=300s
  ```

For non-TLS Enabled Workspaces, this setting is made automatically.

Workaround: Even though an error message displays, project creation still occurs. Check the Projects page after a few minutes; project creation should be complete.

DSE-9549: TLS enabled workspaces require manual configuration

To provision a TLS-enabled workspace, the customer needs to perform several manual steps. This procedure is described in Deploy an ML Workspace with Support for TLS.

OPSAPS-58019: CML workspace installation failure due to includedir in krb5.conf file

If the /etc/krb5.conf on the Cloudera Manager host contains include or includedir directives, Kerberos-related failures may occur.

As a workaround, comment out the include and includedir lines in /etc/krb5.conf on the Cloudera Manager host. If configuration in those files and directories are needed, add them directly to /etc/krb5.conf.

DSE-21768: Spark3 runtime addons are not supported with Python 3.8 Runtimes

If the /etc/krb5.conf on the Cloudera Manager host contains include or includedir directives, Kerberos-related failures may occur.

Spark3 runtime addons are currently not supported with Python 3.8 Runtimes.

Related Information

Known Issues and Limitations