

Machine Learning

Models

Date published: 2020-07-16

Date modified: 2024-06-11

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

Model Training and Deployment Overview.....	5
Models.....	5
Models - Concepts and Terminology.....	6
Challenges with Machine Learning in production.....	8
Challenges with model deployment and serving.....	8
Challenges with model monitoring.....	8
Challenges with model governance.....	10
Model visibility.....	10
Model explainability, interpretability, and reproducibility.....	10
Model governance using Apache Atlas.....	11
Using Model Registry.....	12
Setting up Model Registry.....	13
Creating a Model Registry.....	13
Creating a Model Registry on an Azure UDR Private Cluster.....	15
Setting up access for Model Registry in a RAZ-enabled environment.....	16
Setting up access for Model Registry in a non-RAZ-enabled environment.....	19
Synchronizing the model registry with a workspace.....	19
Viewing Details for Model Registry.....	23
Model Registry permissions.....	23
Model access control.....	24
Deleting Model Registry.....	24
Registering and deploying models with Model Registry.....	24
Creating a model using MLflow.....	24
Registering a model using the Model Registry user interface.....	24
Registering a model using MLflow SDK.....	26
Viewing registered model information.....	26
Creating a new version of a registered model.....	27
Deploying a model from the Model Registry page.....	27
Deploying a model from the destination Project page.....	32
Viewing Details for Model Registry.....	32
Delete a model from Model Registry.....	32
Disabling Model Registry.....	33
Upgrade model registry.....	33
Roll back the registry upgrade.....	33
Model Registry Standalone API.....	34
Prerequisites for Model Registry Standalone API.....	34
Authenticating Clients for interacting with CML Registry API.....	35
Role-based Authorization.....	35
Using the REST Client.....	35
Model Registry CLI Client.....	40
Known issues.....	42
Updating CML Registry configuration.....	43
Troubleshooting issues with Model Registry API.....	43

Creating and Deploying a Model.....	49
Usage Guidelines.....	52
Known Issues and Limitations.....	53
Model Request and Response Formats.....	55
Testing Calls to a Model.....	55
Securing Models.....	57
Access Keys for Models.....	57
API Key for Models.....	57
Enabling authentication.....	58
Generating an API key.....	58
Managing API Keys.....	59
Workflows for Active Models.....	59
Technical Metrics for Models.....	61
Debugging Issues with Models.....	61
Deleting a Model.....	62
Using Cloudera Copilot.....	63
Prerequisites to set up Cloudera Copilot.....	63
Configuring Cloudera Copilot.....	64
Using Cloudera Copilot.....	66
Example - Model Training and Deployment (Iris).....	68
Train the Model.....	69
Deploy the Model.....	71

Model Training and Deployment Overview

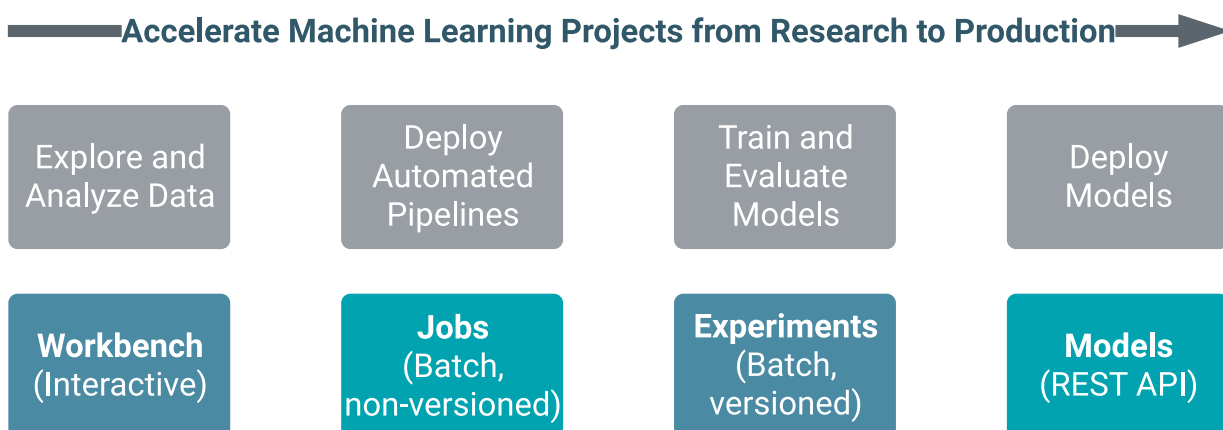
This section provides an overview of model training and deployment using Cloudera Machine Learning.

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.



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:

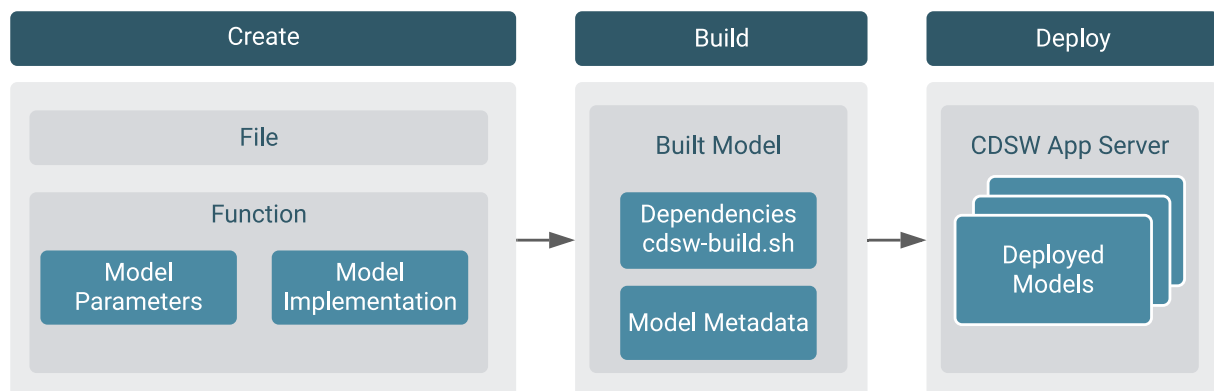
- Create a snapshot of model code, model parameters, and dependencies.
- Package a trained model into an immutable artifact and provide basic serving code.
- Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function's 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

[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 mainly 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

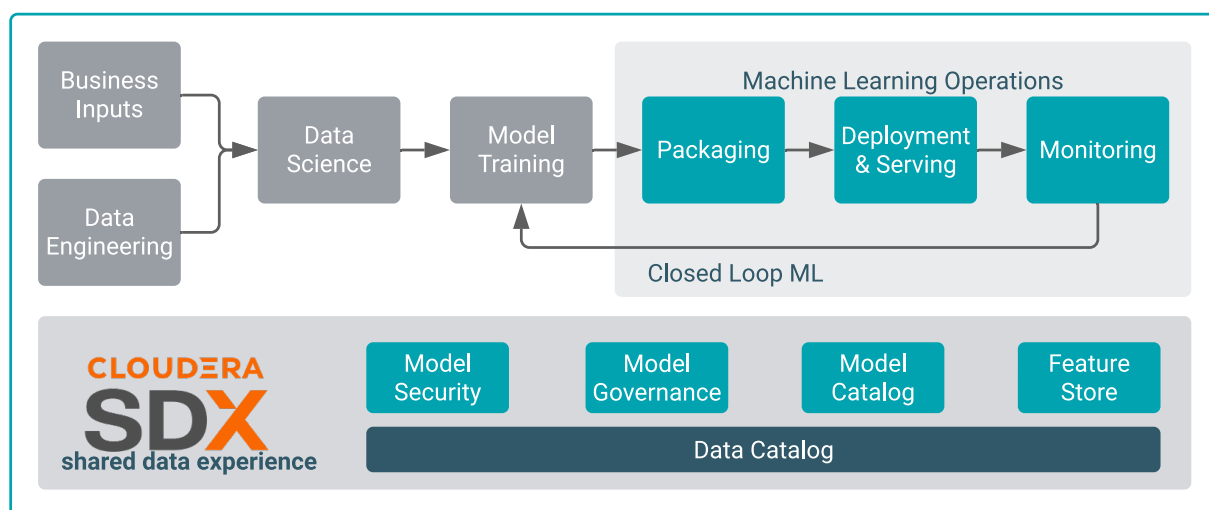
Models are often seen as a black box: data goes in, something happens, and a prediction comes out. This lack of transparency is challenging on a number of levels and is often represented in loosely related terms explainability, interpretability, and reproducibility.

- 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 production ML workflow:

Figure 1: Production ML Workflow



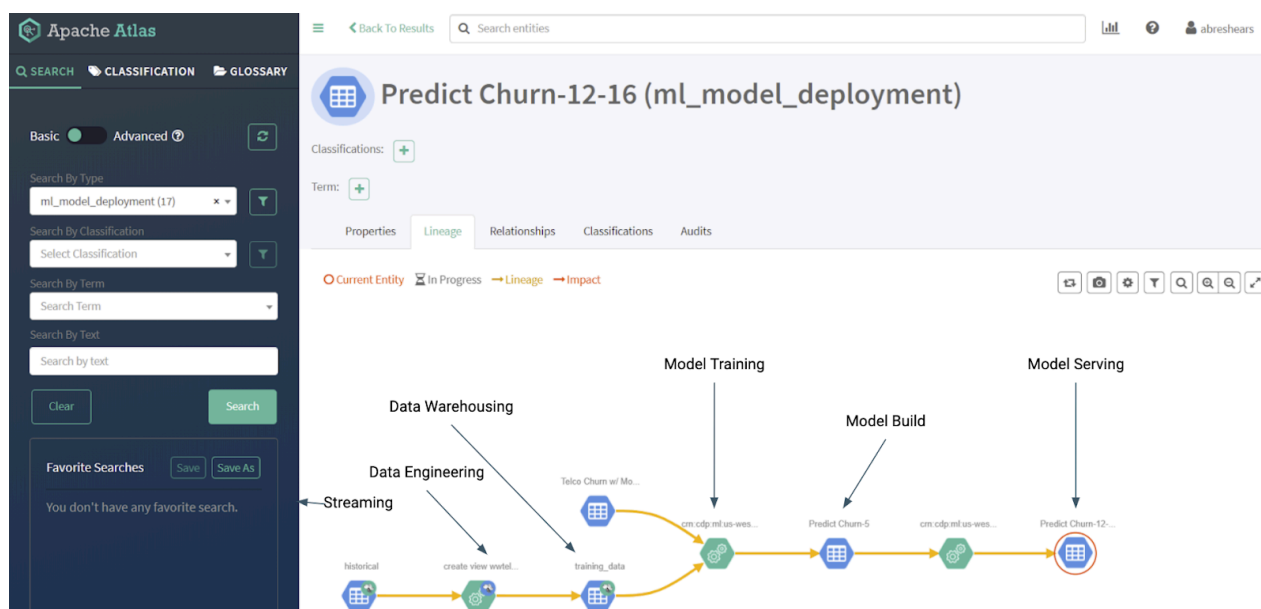
Model governance using Apache Atlas

To address governance challenges, Cloudera Machine 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.

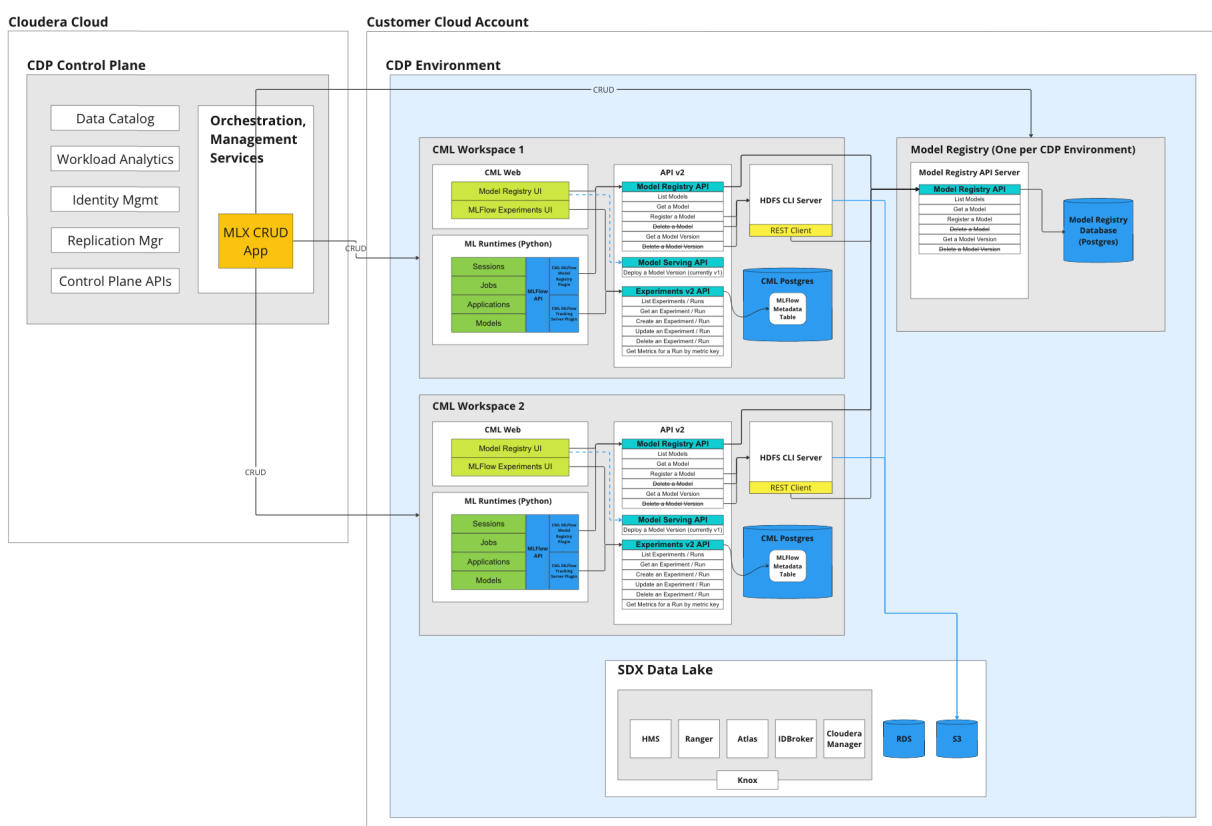
See *ML operations entities created in Atlas* for a list of metadata that Atlas collects from each CML workspace. Metadata is collected from machine learning projects, model builds, and model deployments, and the processes that create these entities.

Using Model Registry

The Model Registry is the core enabler for MLOps, or DevOps for machine learning.

The Model Registry stores and manages machine learning models and associated metadata, such as the model's version, dependencies, and performance. The registry enables MLOps and facilitates the development, deployment, and maintenance of machine learning models in a production environment.

Model Registry in CDP Public Cloud



Model Registry includes functionality for the following tasks:

- Storing and organizing different versions of a machine learning model and its associated metadata.
- Tracking the lineage of a model, including who created it, when it was created, and any changes made to it over time.
- Providing APIs for accessing and deploying models, as well as for querying and searching the registry.
- Integrating with CI/CD pipelines and other tools used in the MLOps workflow.

Model registries help organizations improve the quality and reliability of their machine learning models by providing a centralized location for storing and managing models, as well as enabling traceability and reproducibility of model development. They also make deploying and managing models in a production environment easier by providing a single source for model versions and dependencies.

The Model Registry integrates MLFlow and maintains compatibility with the open source ecosystem.

Limitations

- Upgrade to the General Availability (GA) version of Model Registry might not be supported. Alternatively, upgrade to the GA version of Model Registry might require reinstalling Model Registry which could result in loss of Model Registry data configured with the technical preview (TP) version of Model Registry.

Setting up Model Registry

The Model Registry is the core enabler for MLOps, or DevOps for machine learning.

Prerequisites

- You must have permission to access a project in which the model is created before you can register it.

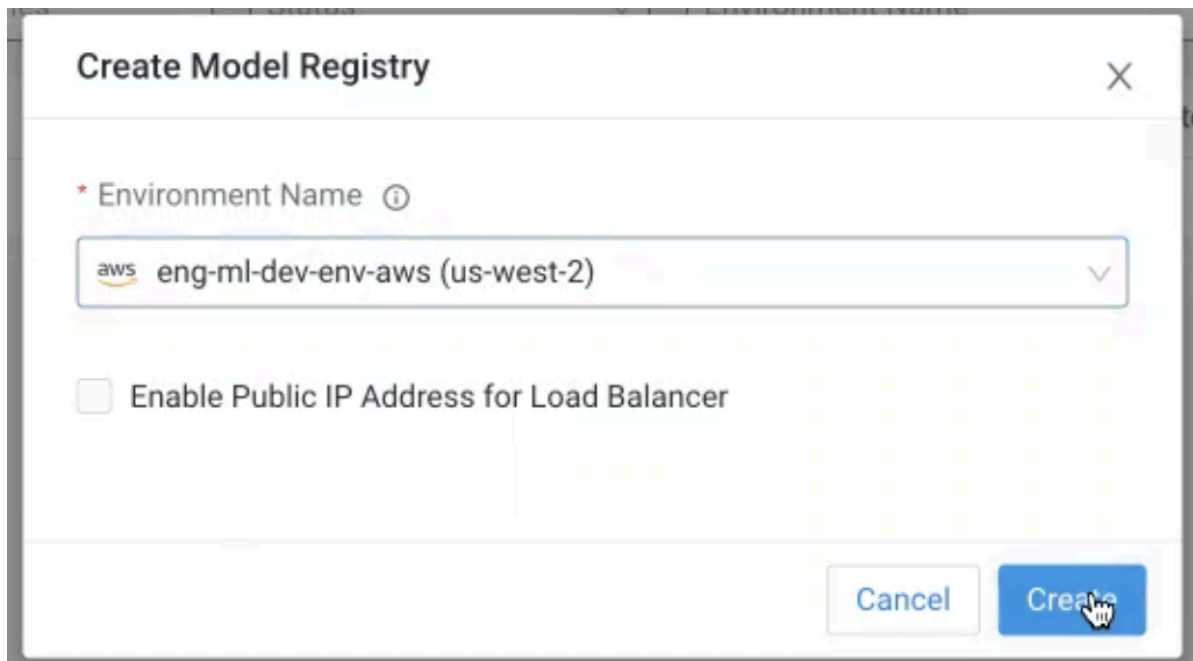
Creating a Model Registry

Before you can start using Model Registry you must create a model registry for your environment.

Procedure

1. From the Cloudera application, click Cloudera Machine Learning.
2. Click Model Registry in the left navigation pane.
3. Choose your environment from the Environment Name drop down list.

4. Depending on your environment, complete one of the following:
- a) If your environment is in AWS, Model Registry displays the following dialog box:



The image shows a 'Create Model Registry' dialog box. At the top, there is a title bar with the text 'Create Model Registry' and a close button (X). Below the title bar, there is a section labeled '* Environment Name' with an information icon (i). Under this label, there is a dropdown menu showing 'aws' as the selected environment and 'eng-ml-dev-env-aws (us-west-2)' as the selected environment name. Below the dropdown menu, there is a checkbox labeled 'Enable Public IP Address for Load Balancer'. At the bottom right of the dialog box, there are two buttons: 'Cancel' and 'Create'. A mouse cursor is pointing at the 'Create' button.

Select your environment, and click Create to create the Model Registry.

- b) If your environment is in Azure, Model Registry displays the following dialog box:

Create Model Registry

* Environment Name ⓘ

dsp-azure-dev (westus2) ▼

* Existing NFS ⓘ

nfs://server:/directory

Note: An administrator must run **chown 8536:8536** on the NFS directory.

The directory must be empty and not used by another workspace.

NFS Protocol version ⓘ

NFS protocol version such as 3 or 4.1 ▼

Cancel Create

1. Choose the Azure environment for the Model Registry.
2. Click Create to create the Model Registry.

Creating a Model Registry on an Azure UDR Private Cluster

Use the following template CDP CLI command to create a UDR private cluster on Azure with a Model Registry.

You must replace the following template items with your own information.

- <environment CRN>
- <environment name> (in two places)
- <subnet>

Model registries are also supported on Azure private clusters with UDR. For more information about UDR, see the [Preview Feature](#) documentation.

If you have not yet downloaded the CDP CLI tool, see the [documentation](#).

Use the latest version of the CDP CLI.

CDP CLI command to create a Model Registry

This CDP CLI command performs has three key sections:

1. Enables support for private clusters in Azure ("privateCluster": true,)
2. Enables UDR for the private cluster ("outboundTypes": ["OUTBOUND_TYPE_UDR"],)
3. Specifies the subnet for the UDR-enabled private cluster ("subnets")

```
cdp ml create-model-registry --cli-input-json {
  "environmentCrn": "<environment CRN>",
  "environmentName": "<environment name>",
  "privateCluster": true,
  "usePublicLoadBalancer": false,
  "outboundTypes": [
    "OUTBOUND_TYPE_UDR"
  ],
  "provisionK8sRequest": {
    "network": {
      "topology": {
        "subnets": [
          "<subnet>" # subnet with a default route configuration to
forward the traffic to the network appliance or firewall. This is required t
o enable UDR.
        ]
      }
    }
  }
}
```

Setting up access for Model Registry in a RAZ-enabled environment

In a RAZ-enabled environment you need to set up the S3-Ranger policy by manually adding the machine user name in the S3 Ranger policy.

To set up the S3-Ranger policy, complete the following:

1. On the Models Registry Details page, find and copy the Machine User Workload User Name in the Machine User Workload User Name field.

For example, in the following screenshot, the Machine User Workload User Name field contains `srv_cml_env_machine_user_82a49`. Copy the Machine User Workload User Name which is 82a49.

Model Registries / model-registry-ml-636f2c89-585

Installing

Details Events & Logs

Name	
Environment Name	eng-ml-dev-env-aws
Environment CRN	crn:cdp:environments:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:...
CRN	crn:cdp:ml:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:workspace:...
Machine User CRN	crn:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:machine...
Machine User Workload User Name	srv_cml_env_machine_user_82a49
Creation Date	04/21/2023 9:17 AM PDT
Creator	crn:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:user:f6fbb5f2-5736-428e-84cd-b236cb3dd04a

2. Go to the Ranger UI in the Datalake of the environment.

Environments / eng-ml-dev-env-aws / Data Lake / Event History

DATA Lake Details

NAME	eng-ml-dev-env-aws-dl	NODES	2 0 0	SCALE	Light Duty	QUICK LINKS	Atlas Ranger Data Catalog
STATUS	Running	STATUS REASON	Datalake cert renewal finished	CRN	crn:cdp:datalake:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:datalake:6a18fa88-0d5e-4f26-95ad-97ffa0145fcb		

Environment Details

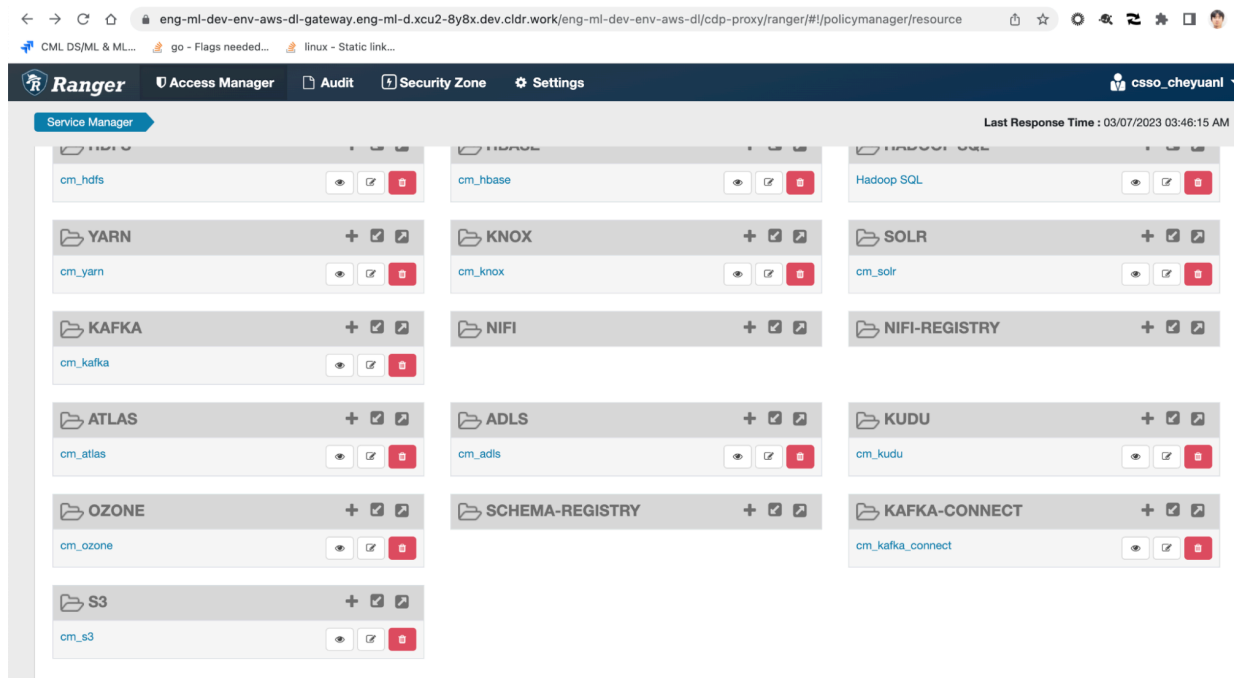
NAME	eng-ml-dev-env-aws	CREDENTIAL	eng-ml-dev-env-aws-creds	REGION	us-west-2	AVAILABILITY ZONE	N/A
------	--------------------	------------	--------------------------	--------	-----------	-------------------	-----

Services

Atlas	CM-UI	HBase UI	Name Node	Ranger
Solo Server	Token Integration			

Cloudera Manager Info

3. Depending on your environment, select `cm_s3` (AWS) or `cm_adls` (Azure).



4. Go to the policy named `all - bucket, path` which controls the access to the object store bucket.

List of Policies : `cm_s3`

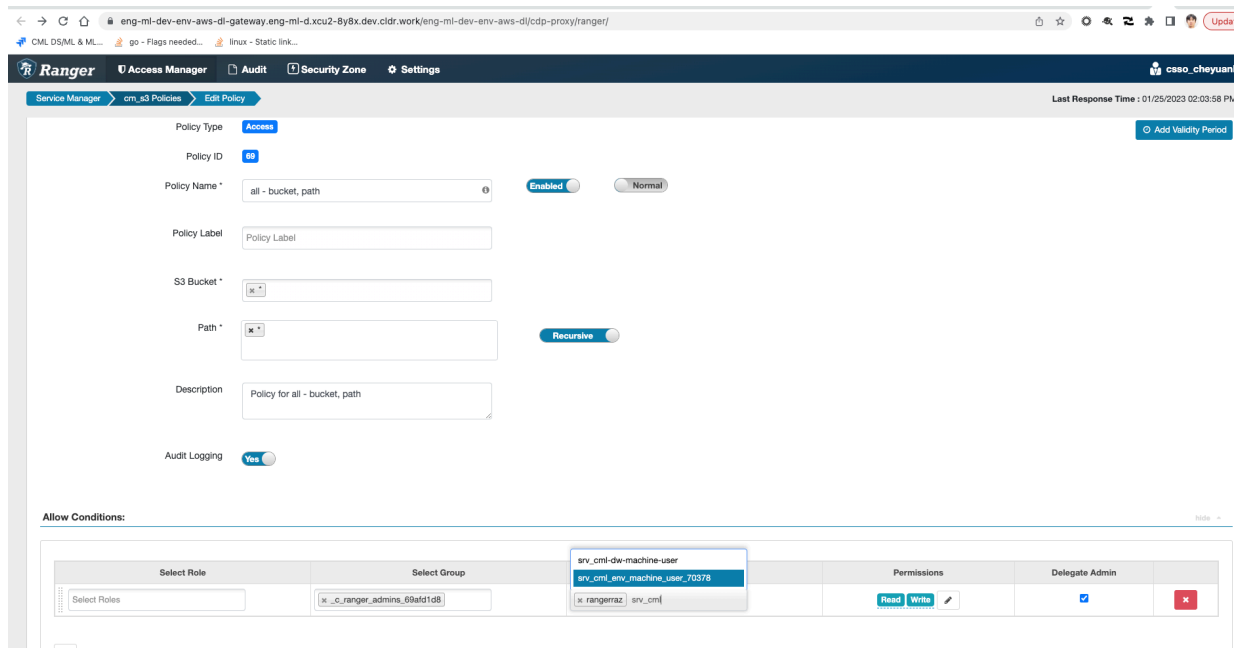
Search for your policy...

Add New Policy

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
69	all - bucket, path	--	Enabled	Enabled	--	<code>_c_ranger_admins_69afd1d8</code>	<code>rangeraz</code> <code>srv_cm_env_machine_user_70378</code>	

5. Enter the Machine User Workload User Name in the Select User field in the allow conditions section.

For example, using the Machine User Workload User Name from Step 2, add the value which is 82a49.

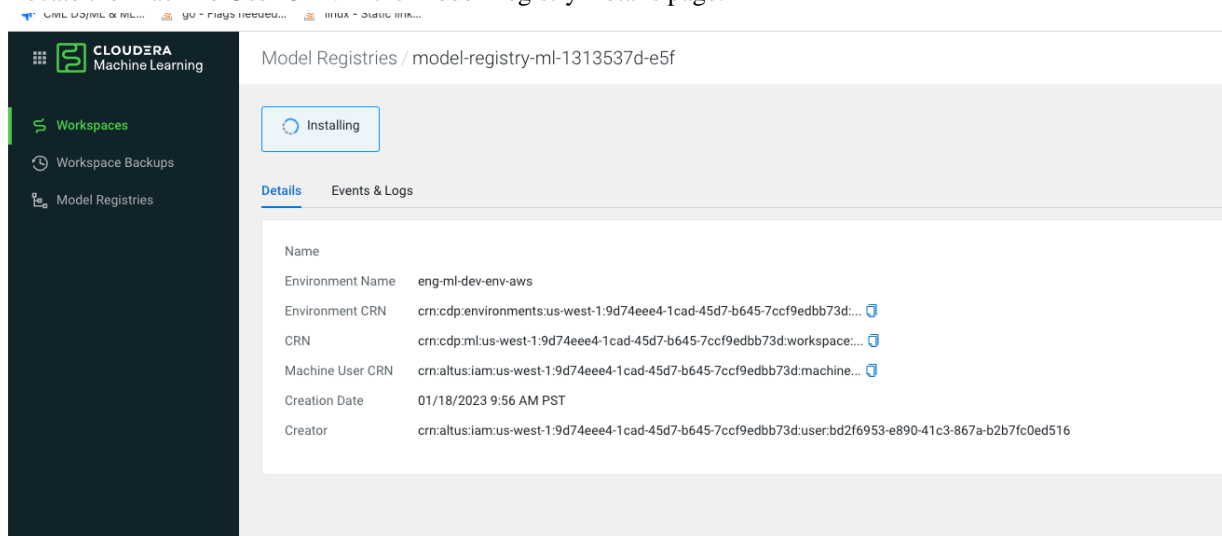


Setting up access for Model Registry in a non-RAZ-enabled environment

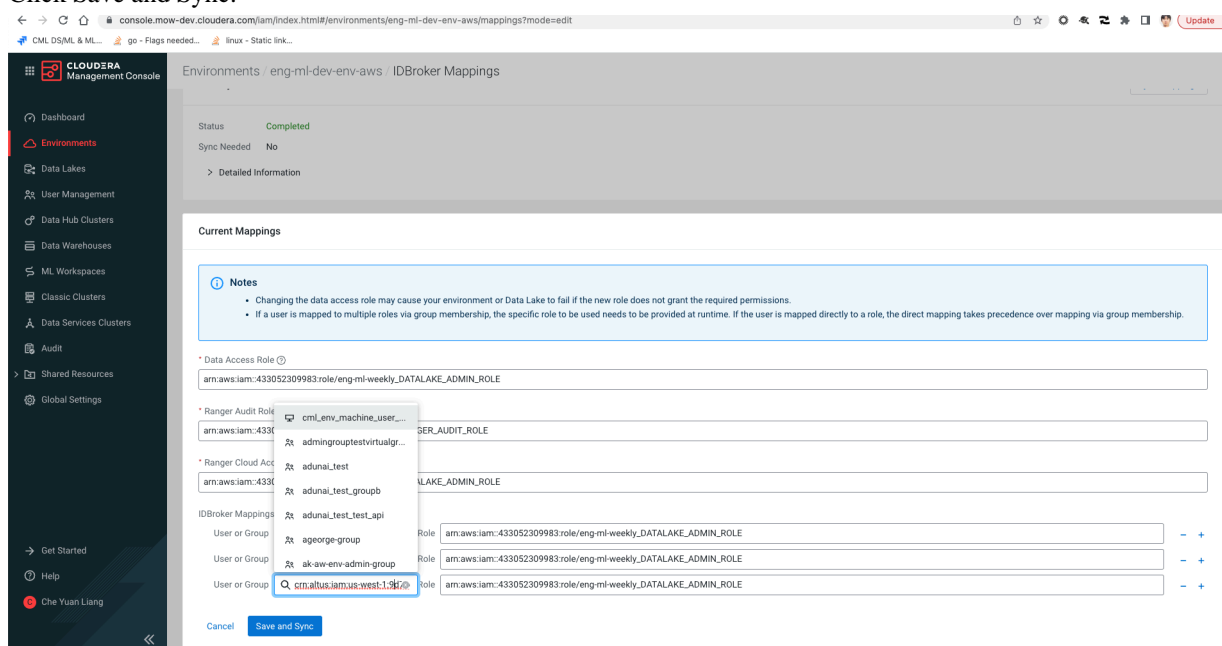
In a non-RAZ-enabled environment you need to add the Machine User CRN to the IDBroker mapping in order to access the S3/ADLS buckets.

To add the Machine User CRN to the IDBroker mapping complete the following:

1. Locate the Machine User CRN in the Model Registry Details page.



2. Copy the entire Machine User CRN mapping.
3. Navigate to the Environment Manage access idbroker page and add or choose the Machine User CRN mapping to the Data Access Role field.
4. Click Save and Sync.



Synchronizing the model registry with a workspace

If you deploy a Model Registry in an environment that contains one or more Cloudera Machine Learning (CML) workspaces, you must synchronize Model Registry with the workspaces.



Important: If your CML Workspace version is 2.0.46 or higher, or Cloudera AI Inference service version is 1.2.0 or higher, and if you deploy a Model Registry in an environment that contains one or more CML Workspaces, the Model Registry is auto-discovered and periodically synchronized by Cloudera AI Inference service and CML Workspaces and no manual synchronization is required.

CML Workspace is auto-synchronized every five minutes and Cloudera AI Inference service is auto-synchronized every 30 seconds.

The outbound network access on [AWS](#) and [Azure](#) must be configured correctly for the auto synchronization to work.

Procedure

1. Click Model Registry to display the Model Registries window.
2. Choose the Model Registry you want to synchronize with the workspaces in the environment.

3. From the Actions menu, click Synchronize.

Model Registry displays the Confirm dialog box listing all of the workspaces in the environment.

⚠ Confirm

Are you sure you want to synchronize the configuration registry with following workspaces?

i Synchronizing will update all CML Workspaces in the CDP environment to use the selected Model Registry. This operation needs to be executed only when a Model Registry is configured in the Environment where CML Workspaces already exist.

Workspace Name

david-test

quasar-cdp-mlx-iwdadk

zoram-serving-exp

4. Click OK.

Viewing Details for Model Registry

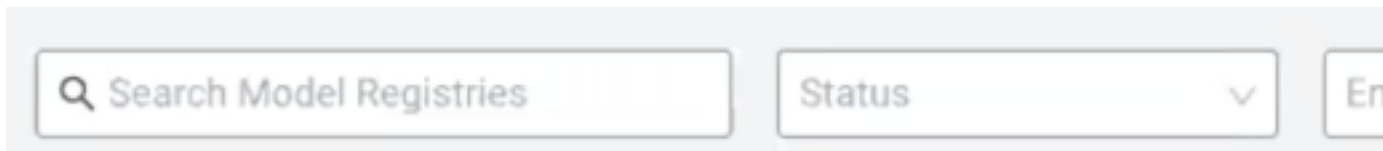
You can view the information for your registered models using Model Registry.

Procedure

1. Select Model Registry from the left navigation pane.

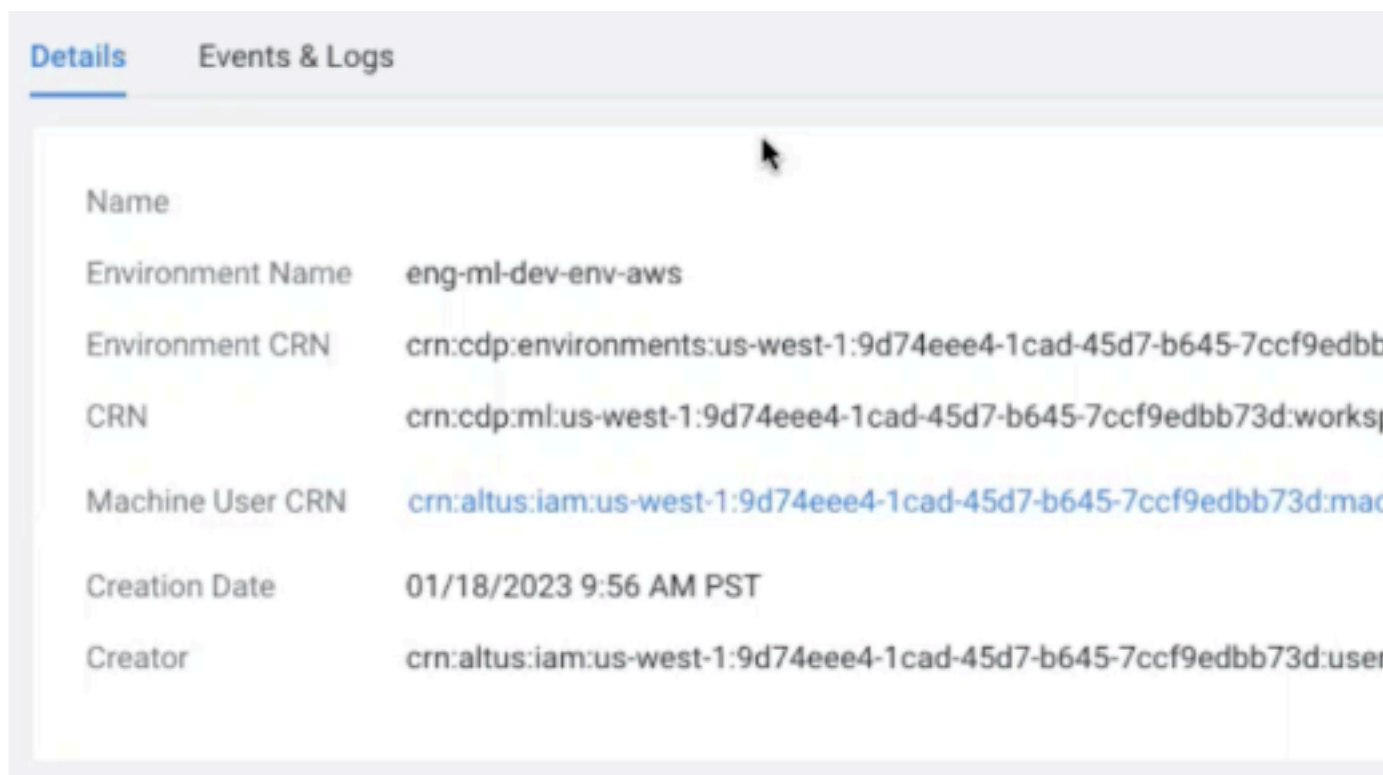
On the main Model Registry page, you can see all the models currently registered, their respective owners, location of creation, and the last updated time, if known.

2. You can use the filter bar at the top of the window to filter the list of model registries by name, status, and environment name.



3. Select a model registry to see its description.

CML displays the Details page which lists the environment name, environment CRN, CRN, machine user CRN, creator, and creation date.



4. You can also click the Events & Logs tab to display information on the events and logs for the model registry.

Model Registry permissions

Model registry permissions for the following actions are separate from workspace permissions, but they are inherited from environment level workspace permissions.

- create
- delete
- getKubeconfig

- grant/list/revoke access

Therefore, if you have the MLAdmin role on an environment, you can perform these actions for model registry, but an MLUser cannot.

Remote access to a model registry works similarly to workspace remote access. In addition to downloading the `kubeconfig` file, you need to use `Grant/List/RevokeModelRegistryAccess` endpoints to manage what cloud user identity can access the Kubernetes cluster using your cloud credential.

Model access control

Access to models is dependent on the user permissions, as described here.

- Only Administrators are able to see all models, including those not created by them.
- A user who is an Owner of a given model can set the public or private visibility of the model through the Model Registry UI.
- When a user registers a new model, that user is the Owner, and no other user has access.
- A user can make a model Public, and then all users receive Viewer access to the model by default.

Deleting Model Registry

If you no longer want to access Model Registry, you can delete it.

Procedure

1. Select Model Registry to display the Model Registries window.
2. Choose the model registry you want to delete.
3. From the Actions menu, choose Delete.
4. Click OK.

Force delete a model registry

If the model registry remains in the Model Registries list and cannot be deleted, you can force delete the model registry with this CDP CLI command:

```
cdp ml delete-model-registry --model-registry-crn <MODEL_REGISTRY_CRN> --force
```

Registering and deploying models with Model Registry

After you've set up Model Registry, you can create, register, and deploy models with Model Registry.

Creating a model using MLflow

You can use MLflow to create a model.

Using MLflow to create a new model

To create a model using MLflow, see <https://docs.cloudera.com/machine-learning/cloud/experiments/topics/ml-exp-v2-mlflow-model-artifact.html>.

Registering a model using the Model Registry user interface

You can register a model using the Model Registry user interface or the MLFlow SDK.

Using the Model Registry user interface to register a model

Registering a model enables you to track your model and upload and share the model. Registering a model stores the model archives in the model registry with a version tag. The first time you register a model, Model Registry automatically creates a model repository with the first version of the model.

Before you begin

You must have permission to access a project in which the model is created before you can register the model.

Procedure

1. Click Projects in the left navigation pane to display the Projects page.
2. Select the project that contains the model that you want to register.
CML displays all of the models under the specific project along with their source, deployment status, replicas, memory and a drop-down function for actions that can be made pertaining to that model for deployment.
3. Click the Experiments tab in the left navigation pane and select the experiment that contains the model you want to register.
4. Select the model you want to register.
CML displays the Experiment Run Details page.

The screenshot shows the Cloudera Machine Learning (CML) Model Registry user interface. The left navigation pane includes options like All Projects, Overview, Sessions, Experiments, Model Deployments, Jobs, Applications, Files, Collaborators, and Project Settings. The main content area displays the 'admin / test1 / Experiments / registermodeltest / Run' page. It features several sections: Metrics (mse, r2, mae), Parameters (alpha, l1_ratio), Tags (No Data), and Artifacts (requirements.txt, conda.yaml, model.pkl, python_env.yaml, MLmodel). The 'Make Predictions' section provides code snippets for predicting on a Spark DataFrame and a Pandas DataFrame.

5. Select the run that contains the model you want to register.

6. Select Register Model to begin the registration process.
Model Registry displays the Registry Model dialog box.
7. Enter the name of your registered model.
You can also enter optional information for the description, version notes, and version tags.
8. Click OK to complete the registration.

Registering a model using MLflow SDK

You can register a model using the user interface or the MLFlow SDK.

Using MLflow SDK to register a model

Registering a model enables you to track your model and upload and share the model. Registering a model stores the model archives in the model registry with a version tag. The first time you register a model, Model Registry automatically creates a model repository with the first version of the model.

Procedure

1. To register a model using MLFlow SDK, specify the `registered_model_name` and assign a value:

```
mlflow.<model_flavor>.log_model()
```

For example:

```
mlflow.sklearn.log_model(lr, "model", registered_model_name="ElasticnetW  
ineModel")
```

2. If you run the Python code again with the same `model_name` it will create an additional version for the `model_name`.

Using MLflow SDK to register customized models

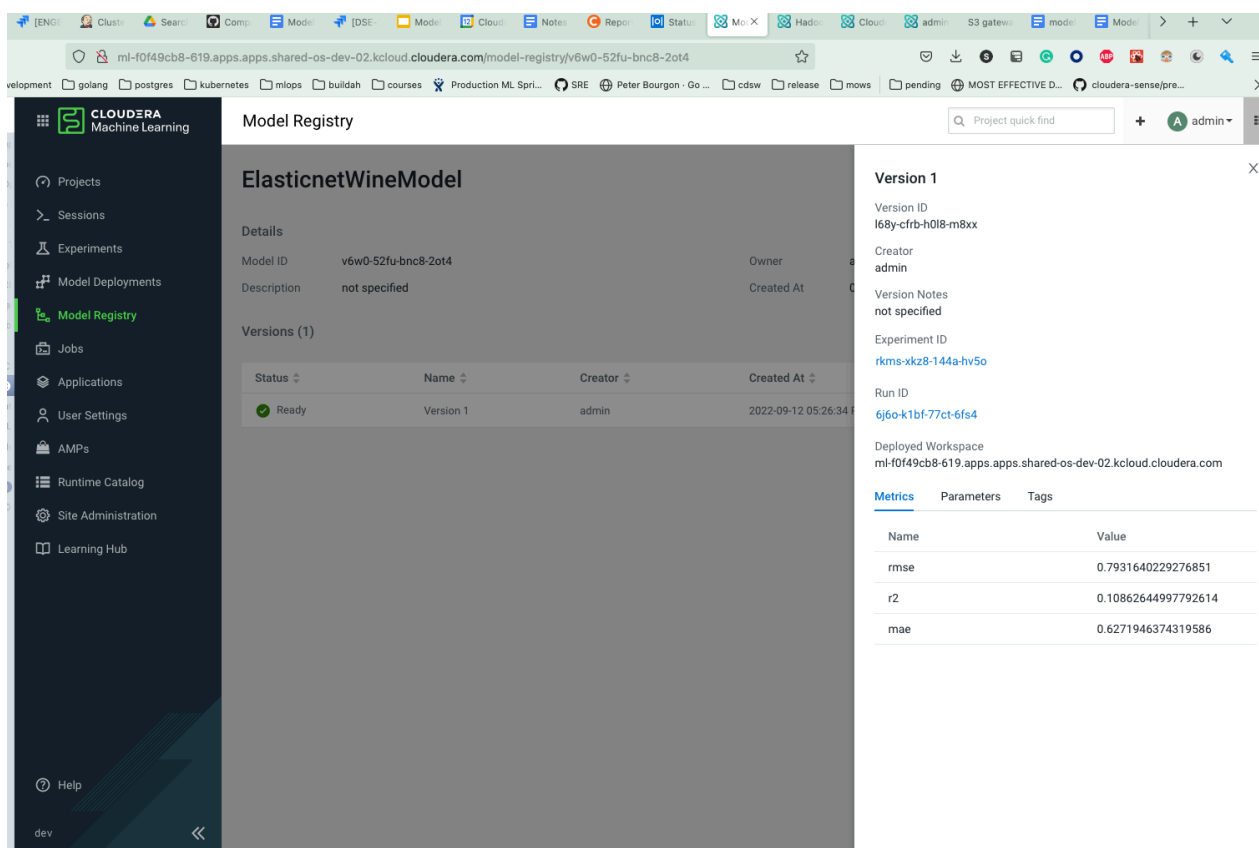
In MLflow, you can also deploy models that are not directly supported by MLFlow.

To learn more, see [Serving LLMs with MLflow: Leveraging Custom PyFunc](#).

Viewing registered model information

1. From the Projects page in CML, select Model Registry from the navigation pane. On the main Model Registry page, you can see all the models currently registered, their respective owners, location of creation, and the last updated time, if known.
2. Select a registered model to see its description.

CML displays the Details page which outlines the model description, ID, owner, and versions. Different versions of the same model can be deployed in the workspace.



Creating a new version of a registered model

You can easily create a new version of a registered model.

Procedure

1. Click Projects in the left navigation pane to display the Projects page.
2. Select the project that contains the model for which you want to create a new version.
3. Click Experiments in the left navigation pane and select the experiment that contains the model you want to register.

The system displays the Experiment Detail page.

4. Select the run that contains the model you want to register.
5. Scroll down the page to find the Artifacts section and click model.
6. Click Register Model.
7. From the Name field, choose the model for which you want to create a new version.
8. Click OK.

What to do next

You can also create a new model version using MLflow SDK. Simply run the Python code to register a model again with the same model_name. This will create an additional version for the model_name.

Deploying a model from the Model Registry page

You can deploy a model one or more times to create different versions of the model. You can also deploy a model you created in one workspace to a different workspace.

Procedure

1. Select Model Registry from the left navigation pane.
2. Select the model you want to deploy.
Model Registry display the Model Version List page.
3. Select the model version you want to deploy.
Model Registry displays a side window that lists the version information. Dismiss this window to proceed.
4. Under the Actions menu, click Deploy.

5. Select the Project you want to deploy to in the dialog box and click Go.

You can select either the project the model is located in or another project to deploy the model to.

Model Registry displays the Deploy a Model page with the detailed model information auto populated.

The screenshot shows the Cloudera Machine Learning web interface. The browser address bar displays the URL `ml-6acb04c2-8f2.apps.apps.shared-os-dev-02.kcloud.cloudera.com`. The breadcrumb navigation at the top reads `admin / test3 / Model Deployments`. The left sidebar contains a navigation menu with the following items: All Projects, Overview, Sessions, Experiments, **Model Deployments** (highlighted in green), Jobs, Applications, Files, Collaborators, and Project Settings. The main content area is titled **Deploy a Model**. Under the **Deployment Template** section, there are two radio buttons: ☐ Deploy model from code and ☒ Deploy registered model. The **General** section contains three fields: *** Registered Model** with the value `ElasticnetWineModel`, *** Model Version** with the value `Version 1`, and a checked checkbox for **Enable Authentication**. A blue information box below the checkbox states: **i** Enforces model API requests to be authenticated.

development golang postgres kubernetes mlops buildah courses

CLOUDERA
Machine Learning

← All Projects

Overview

Sessions

Experiments

Model Deployments

Jobs

Applications

Files

Collaborators

Project Settings

admin / test3 / Model Deployments

Deploy a Model

Deployment Template

☐ Deploy model from code

☒ Deploy registered model

General

*** Registered Model**

ElasticnetWineModel

*** Model Version**

Version 1

☒ **Enable Authentication**

i Enforces model API requests to be authenticated

Build

6. If you enable authentication, you will need to enter an API key to access and use the model in the case you have deployed the model to a shared project.
7. Click OK.

Deploying a model from the Model Registry using APIv2

You can use the API v2 to deploy registered models from the Model Registry as part of your MLOps CI/CD pipeline.

The following example code shows how to deploy a model from the Model Registry by using three APIv2 calls: create a model, create a model build, and create a model deployment.

```
api_client = cmlapi.default_client()

model_body = cmlapi.CreateModelRequest(
    project_id=project_id,
    name="foo", # replace this with the model
    name
    description="Foo",
    disable_authentication=True,
    registered_model_id="xyo2-ohbr-w0n2-dx3s" # replace this with the register
ed model id
)

model = api_client.create_model(model_body, project_id)
print(model)

model_build_body = cmlapi.CreateModelBuildRequest(
    project_id=project_id,
    model_id=model.id,
    kernel="python3",
    runtime_identfier='docker.repository.cloudera.com/cloudera/cdsw/ml-run
time-pbj-workbench-python3.10-standard:2023.12.1-b8', # replace this with th
e runtime identifier
    registered_model_version_id="ar0a-z7sd-pjgb-2fn2" # replace this with the
registered model id
)

model_build = api_client.create_model_build(model_build_body, project_id, mo
del.id)
print(model_build)

while model_build.status not in ["built", "build failed"]:
    print("waiting for model to build...")
    time.sleep(10)
    model_build = api_client.get_model_build(project_id, model.id, model_b
uild.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=proj
ect_id, model_id=model.id, build_id=model_build.id, replicas = model_replica
s)
model_deployment = api_client.create_model_deployment(model_deployment_bo
dy, project_id, model.id, model_build.id)

while model_deployment.status not in ["stopped", "failed", "deployed"]:
    print("waiting for model to deploy...")
    time.sleep(10)
    model_deployment = api_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!")
```

Deploying a model from the destination Project page

You can deploy a model one or more times to create different versions of the model. You can also deploy a model you created in one workspace to a different workspace.

Procedure

1. Navigate to the Project you want to deploy to.
2. Click Model Deployment in the left navigation pane.
3. Make sure you have clicked the Deploy registered model checkbox at the top of the window.
4. Select the registered model you want to deploy from the Deploy Registered Model field.
5. If you enable authentication, you will need to enter an API key to access and use the model in the case you have deployed the model to a shared project.
6. Select Deploy Model at the bottom of the window.

Viewing Details for Model Registry

You can view detailed information for Model Registry.

Procedure

1. Select Model Registry from the left navigation pane.
On the main Model Registry page, you can see all the models currently registered, their environment name, respective owners, location of creation, and the last updated time, if known.
2. Choose the model registry you want to synchronize with the workspaces in the environment.
3. You can use the filter bar at the top of the window to filter the list of model registries by name, status, and environment name.
4. Click OK.

Delete a model from Model Registry

You can delete a model from Model Registry through the UI or by means of an API call.

Deleting through the UI

1. In Model Registry, find the model to delete.
2. In Actions, select Delete.
3. Click OK to confirm deleting the model.

The model is deleted from the Model Registry.

Deleting a model with an API call

You can run API calls in the session workbench to delete a model.

1. Use the first two commands to obtain the model_id:

```
api_client=cmlapi.default_client()
api_client.list_registered_models()
```

The json output of the command includes an example model_id as shown here:

```
'model_id': '7xwf-e6pl-tb28-iy1h',
```


2. Insert the `model_id` (replace the example shown below with your own value) to the following command and run it.

```
api_client.delete_registered_model(model_id='7xwf-e6pl-tb28-iy1h')
```

The model is deleted from the Model Registry.

Disabling Model Registry

By default, Model Registry is enabled in CML. You can disable Model Registry if you do not want to use this feature.

Procedure

1. Click Site Administration in the left navigation pane.
2. Click Settings to display the Setting Page.
3. Under the Feature Flags section, uncheck the Enable Model Registry checkbox.

Upgrade model registry

When you upgrade a model registry, you create a new version of the registry. You can roll back to the old version, or delete the old registry if it is no longer needed.

The Model Registry upgrade process involves provisioning a new cluster, so the process can take some time, on the order of 40 to 60 minutes. The upgrade operation should be performed during a maintenance window, when new models are not being added or updated in CML. The window should be approximately 3 hours long at a minimum.

1. In Model Registries, choose a registry to upgrade.
2. Copy the CRN (Creator) shown in the Model Registry.
3. In the CDP CLI, enter `cdp ml upgrade-model-registry --crn <crn>`

In the Model Registries UI, the Status of the model registry changes to Backup Initiated. The process starts provisioning a new registry (Status: Creating) and then performs a backup operation to restore the state.

4. When the new registry shows Ready, then in Actions, click Synchronize. Check the contents of the new registry to verify that all of the models have transferred.
5. Finally, after verifying the contents of the new registry, you can delete the old registry (check the Created at timestamp) in the Action menu. This action is irreversible.

To verify the data, go to a workspace. Check that all of the models are there. Deleting the old model registry cannot be reversed.

You should also check the event logs to see if any errors occurred during the upgrade. Can also be used real time to see that the process is running. If there is a need for an escalation ticket, make sure to include the logs.

If you upgrade a model registry, then synchronize, then rollback, any models that were added through synchronizing will not be present. The process doesn't transfer the data backwards to the old registry.

Roll back the registry upgrade

If there is a problem, you can roll back the upgrade using the endpoint: `rollback-model-registry-upgrade`

At the command line prompt, enter the following command:

```
cdp ml rollback-model-upgrade-registry --crn <crn>
```

For `crn`, use the `crn` of the registry you upgraded (that is, the same `crn` you used in the upgrade command)

For more information, you can check the help document for these commands:

- `cdp ml rollback-model-upgrade-registry help`
- `cdp ml upgrade-model-registry help`

Model Registry Standalone API

You can use standalone CML registry API to communicate with the model registry using the REST client or CLI client.

After the release of CML Registry General Availability (GA), the CML Registry API has been exposed through CML workspace APIv2. So, a CML workspace must be present in the same CDP environment to communicate with the CML registry. For more information, see [REST API](#).

The CML Registry Standalone API supports the following functionalities:

- GET/PATCH/DELETE for the model and model version
- GET a curated list of NGC models
- Import external model from [NVIDIA NGC](#) or [HuggingFace](#) to CML Registry through the POST method

Currently, the CML Registry Standalone API does not support uploading the models through POST method from the local machine.

Cloud Platforms

CML Registry API is available only on AWS and Azure.

API definition

The Swagger definition is available in the [CML API documentation](#).

Prerequisites for Model Registry Standalone API

To set up the Cloudera Machine Learning Registry standalone API, configure the Cloudera AI Inference and import pretrained Models.

Prerequisites for Cloudera AI Inference

CML Registry is a prerequisite for Cloudera AI Inference because the AI inference service needs to deploy the models that are stored in the CML Registry.

- To use the Cloudera AI Inference service, the latest CML Registry must be present in the same CDP environment before the Cloudera AI Inference service is created.
- If there is an older CML Registry in the environment that is created before May 14, 2024, follow the *Upgrade Model Registry* instructions to upgrade the Registry to the latest version before you create the Cloudera AI Inference service.
- If the CML Registry is re-created, upgraded, or cert-renewed while the Cloudera AI Inference is present, then follow the steps listed in the *Manually Updating Model Registry Configuration* topic to ensure that the configuration of CML Registry and Cloudera AI Inference are synchronized.

Prerequisites to import pretrained models

You must add the URL details to allow them in the firewall rules.

NVIDIA GPU Cloud (NGC)

Add the following URL details so they can be allowed in the firewall's rules.

- prod.otel.kaizen.nvidia.com (NVIDIA open telemetry)
- api.ngc.nvidia.com
- files.ngc.nvidia.com

Hugging Face

Add the following URL details so they can be allowed in the firewall's rules.

- huggingface.co

- cdn-lfs.huggingface.co
- *.cloudfront.net (CDN)



Note: If required, you must allow more URLs based on your requirements.

Authenticating Clients for interacting with CML Registry API

Clients that interact with the CML Registry Standalone API and with model endpoints must obtain a JSON Web Token (JWT) from the CDP control plane, which must be passed as a Bearer token in HTTP requests sent to the serving API and endpoints.

To obtain JWT, run the following CDP CLI command:

```
$ CDP_TOKEN=$(cdp iam generate-workload-auth-token --workload-name DE | jq -r '.token')
```

In this comment, *DE* is the workload name.

Then pass CDP_TOKEN in the HTTP request header as follows

```
$ curl -H "Authorization: Bearer ${CDP_TOKEN}" <URL>
```

The token obtained using this method expires in one hour.

Role-based Authorization

CML Registry implements role-based access control. It allows access for users with the following IAM roles:

- MLUser
- MLAdmin (admin user)

For more information about the access control for the registered models, see *Model access control*.

Related Information

[Model access control](#)

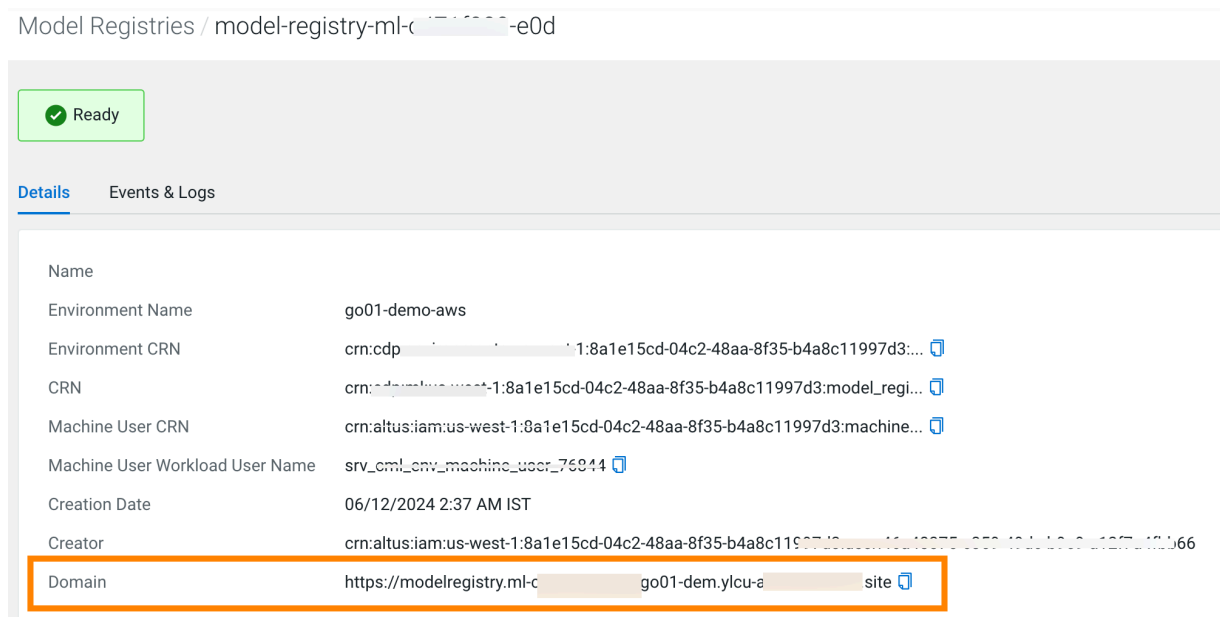
Using the REST Client

You need the domain information to use the REST client to interact with the registry.

To obtain the domain information, perform the following:

1. In the **Cloudera Data Platform** (CDP) console, click the **Machine Learning** tile.
2. Click **Model Registries** in the left navigation menu. The **Model Registries** page displays.

3. Click on the name of the model registry to display the model registry information. The Domain name is displayed in the Details tab.



Importing external models from NGC Catalog to CML Registry

To use the CML Registry standalone API, the CML registry must have access to the public internet to download the model to the CDP environment Data Lake.

Before you begin

You must configure the firewall to allow traffic.

Get the curated list of NGC models

The models listed in the response are curated by Cloudera, those models are pre-compiled and optimized for the specific GPU hardware used by Cloudera AI Inference Service. The `repo_id` is the identifier for the model import input.

```
curl -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/modelhub/models

{
  "models": [
    {
      "application": "Other",
      "framework": "NeMo",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/llama-2-13b:LLAMA-2-13B-4K-FP16-1-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/llama-2-70b-chat:LLAMA-2-70B-CHAT-4K-FP16-4-A100.24.01",
      "source": "NGC"
    }
  ]
}
```

```

    {
      "application": "Other",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/llama-2-13b-chat:LLAMA-2-13B-
CHAT-4K-FP16-1-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "NeMo",
      "name": "Llama-2-70b",
      "precision": "FP16",
      "repo_id": "ohlfw0laadg/ea-participants/mixtral-8x7b:MIXTRAL-8x7b-I
NSTRUCT-2-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/nv-gpt-43b-chat:NV-GPT-43B-
chat-4K-steerlm-FP16-2-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/nv-gpt-43b-instruct:NV-GPT-4
3B-instruct-4K-sft-FP16-2-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "NeMo",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/starcoderplus:STARCODERPLUS
-15.5B-8K-FP16-1-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "Other",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "FP16",
      "repo_id": "ohlfw0laadg/ea-participants/mistral-7b-instruct:MISTRA
L-7b-INSTRUCT-1-A100.24.01",
      "source": "NGC"
    },
    {
      "application": "NATURAL_LANGUAGE_PROCESSING",
      "framework": "Other",
      "name": "Llama-2-70b",
      "precision": "float16",
      "repo_id": "ohlfw0laadg/ea-participants/starcoder:STARCODER-15.5B-8K-
FP16-1-A100.24.01",
      "source": "NGC"
    },
  ],
  {

```

```

    "application": "NATURAL_LANGUAGE_PROCESSING",
    "framework": "NeMo",
    "name": "Llama-2-70b",
    "precision": "float16",
    "repo_id": "ohlfw0laadg/ea-participants/llama-2-7b:LLAMA-2-7B-4K-FP16-1-A100.24.01",
    "source": "NGC"
  },
  {
    "application": "Other",
    "framework": "Other",
    "name": "Llama-2-70b",
    "precision": "float16",
    "repo_id": "ohlfw0laadg/ea-participants/nv-gpt-8b-base:NV-GPT-8B-base-4K-FP16-1-A100.24.01",
    "source": "NGC"
  },
  {
    "application": "Other",
    "framework": "Other",
    "name": "Llama-2-70b",
    "precision": "float16",
    "repo_id": "ohlfw0laadg/ea-participants/llama-2-7b-chat:LLAMA-2-7B-CHAT-4K-FP16-1-A100.24.01",
    "source": "NGC"
  },
  {
    "application": "Other",
    "framework": "Other",
    "name": "Llama-2-70b",
    "precision": "float16",
    "repo_id": "ohlfw0laadg/ea-participants/llama-2-70b:LLAMA-2-70B-4K-FP16-4-A100.24.01",
    "source": "NGC"
  }
]
}

```

Import the selected model to CML Registry

```

curl -XPOST -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/models -d '{
  "name": "llama",
  "createModelVersionRequestPayload": {
    "metadata": {
      "model_repo_type": "NGC"
    },
    "downloadModelRepoRequest": {
      "source": "NGC",
      "repo_id": "ohlfw0laadg/ea-participants/nv-embed-qa:4_hf"
    }
  }
}'

```

```

curl -XPOST -H "Content-Type: application/json" -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/models -d '{
  "name": "llama",
  "createModelVersionRequestPayload": {
    "metadata": {
      "model_repo_type": "NGC"
    },
    "downloadModelRepoRequest": {

```

```

        "source": "NGC",
        "repo_id": "ohlw00laadg/ea-participants/llama-2-7b-
chat:LLAMA-2-7B-CHAT-4K-FP16-1-A100.24.01"
    }
}

```

Importing external models from HuggingFace

You can get the HuggingFace model version identifier from HuggingFace.

Before you begin

To discover the model, see [Hugging Face Models](#).

Import the selected model to CML Registry

```

curl -XPOST -H "Content-Type: application/json" -H "Authorization: Bearer ${
CDP_TOKEN}" "${DOMAIN}/api/v2/models" -d '{
  "name": "tiny",
  "createModelVersionRequestPayload": {
    "metadata": {
      "model_repo_type": "HF"
    },
    "downloadModelRepoRequest": {
      "source": "HF",
      "repo_id": "prajjwall/bert-tiny"
    }
  }
}'

```

Get all models

```

curl -s -H "Authorization: Bearer ${CDP_TOKEN}" "${DOMAIN}/api/v2/models" | jq
{
  "models": [
    {
      "created_at": "2024-04-18T15:54:15.543Z",
      "creator": {
        "user_name": "csso_cheyuanl"
      },
      "id": "5bwt-qqe2-elvg-chqj",
      "name": "foo",
      "tags": null,
      "updated_at": "2024-04-18T15:54:15.543Z",
      "visibility": "private"
    },
  ],
}

```

Get a model

```

curl -s -H "Authorization: Bearer ${CDP_TOKEN}" "${DOMAIN}/api/v2/models/fx0k
-baf7-ysz1-jrt2" | jq
{
  "created_at": "2024-04-18T15:54:24.940Z",
  "creator": {
    "user_name": "csso_cheyuanl"
  },
  "id": "fx0k-baf7-ysz1-jrt2",
  "model_versions": [
    {

```

```

    "artifact_uri": "abfs://data@engmldevenvazuresan.dfs.core.windows.net/
modelregistry/fx0k-baf7-ysz1-jrt2/y8d8-qluc-00md-h2pw/model.tar.gz",
    "created_at": "2024-04-18T15:54:24.942Z",
    "model_id": "fx0k-baf7-ysz1-jrt2",
    "status": "READY",
    "tags": null,
    "updated_at": "2024-04-18T15:54:24.942Z",
    "user": {
      "user_name": "csso_cheyuan1"
    },
    "version": 1
  }
],
"name": "foo2",
"tags": null,
"updated_at": "2024-04-18T15:54:24.940Z",
"visibility": "private"
}

```

Get a model version

```

curl -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/models/fx0k
-baf7-ysz1-jrt2/versions/1 | jq
{
  "artifact_uri": "abfs://data@engmldevenvazuresan.dfs.core.windows.net/mo
delregistry/fx0k-baf7-ysz1-jrt2/y8d8-qluc-00md-h2pw/model.tar.gz",
  "created_at": "2024-04-18T15:54:24.942Z",
  "model_id": "fx0k-baf7-ysz1-jrt2",
  "status": "READY",
  "tags": null,
  "updated_at": "2024-04-18T15:54:24.942Z",
  "user": {
    "user_name": "csso_cheyuan1"
  },
  "version": 1
}

```

Delete a Model

```

curl -XDELETE -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/mo
dels/vuu6-gcfx-ydio-rit0

```

Delete a Model Version

```

curl -XDELETE -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/mo
dels/vuu6-gcfx-ydio-rit0/versions/1

```



Note: The Patch Model and Patch Model API are not supported.

Model Registry CLI Client

Model registry client is a command line tool (CLI) that can be used to interact to a model registry server. It can be downloaded from any model registry server `<domain>/apiv2/cli/<os>`. Here, `<os>` is either Linux, Darwin for Mac, or Windows based on the operating system the model registry CLI is installed on.

The swagger CLI is downloaded from `https://<domain>/apiv2/cli/<os>`. The following are some of the example usage of CLI.

Usage

After you download the CLI and add it to the path, you can use the `modelregistrycli` commands.

```
modelregistrycli help
Usage:
  modelregistrycli [command]

Available Commands:
  completion  Generate completion script
  help        Help about any command
  operations

Flags:
  --Authorization string      config file path
  --config string             output debug logs
  --debug                    do not send the request to server
  --dry-run                  help for modelregistrycli
  -h, --help                 hostname of the service (default "localhost")
  --hostname string           Choose from: [http] (default "http")
  --scheme string
```

Create a Model

Create an imported model request

```
$ modelregistrycli --Authorization "Bearer nil" --hostname localhost:8188 op
erations CreateModel --body '{
  "name": "tiny",
  "createModelVersionRequestPayload": {
    "metadata": {
      "model_repo_type": "HF"
    },
    "downloadModelRepoRequest": {
      "source": "HF",
      "repo_id": "prajjwall/bert-tiny"
    }
  }
}'
***Output***

{"created_at":"2024-04-03T18:02:15.331Z","creator":{"user_name":"admin"},
"description":"model to classify catAndDogClassifier","id":"1w6s-8m6t-ngdr-3
qvr","model_versions":null,"name":"catAndDogClassifier","tags":[{"key":"cat"
,"value":"1"}],"updated_at":"2024-04-03T18:02:15.331Z","visibility":"public"
}
```

Get Models

```
$ modelregistrycli --Authorization "Bearer nil" --hostname localhost:8188 op
erations GetModels

***output***:

{"models":[{"created_at":"2024-04-03T18:02:15.331Z","creator":{"user_name
":"admin"},"description":"model to classify catAndDogClassifier","id":"1w6s-
8m6t-ngdr-3qvr","model_versions":null,"name":"catAndDogClassifier","tags":nu
ll,"updated_at":"2024-04-03T18:02:15.331Z","visibility":"public"},{"created_
at":"2024-04-03T18:08:43.130Z","creator":{"user_name":"admin"},"description"
:"create request model request with model version example","id":"8fts-rgpn-r
9xo-xlh0","model_versions":null,"name":"chain-classifier","tags":null,"updat
ed_at":"2024-04-03T18:08:43.130Z","visibility":"public"}]}
```

Get Model by ID

```
$ modelregistrycli --Authorization "Bearer nil" --hostname localhost:8188 operations GetModel --model_id '8fts-rgpn-r9xo-xlh0'
{"created_at": "2024-04-03T18:08:43.130Z", "creator": {"user_name": "admin"}, "description": "create request model request with model version example", "id": "8fts-rgpn-r9xo-xlh0", "model_versions": [{"artifact_uri": "http://localhost:9000/8fts-rgpn-r9xo-xlh0/r5eg-m0gp-i4qs-8b07/model.tar.gz", "created_at": "2024-04-03T18:08:43.132Z", "model_id": "8fts-rgpn-r9xo-xlh0", "notes": "create request model request with model version example", "status": "REGISTERING", "tags": [{"key": "chain", "value": "2"}], "updated_at": "2024-04-03T18:08:43.132Z", "user": {"user_name": "admin"}, "version": 1}], "name": "chain-classifier", "tags": null, "updated_at": "2024-04-03T18:08:43.130Z", "visibility": "public"}
```

Known issues

These are some of the known issues you might run into while using Model Registry Standalone API.

NGC model download timeout

The NGC model import might time out, and the corresponding model version status is shown as “failed”. You can access the logs found in the API v2 pod by performing the steps mentioned in the *Debugging the model import failure* troubleshooting section.

```
2024/04/23 16:53:45 Error download model repo: ohlfw00laadg/ea-participants/1l
7B-CHAT-4K-FP16-1-A100.24.01
2024/04/23 16:53:45 Error: exit status 1
2024/04/23 16:53:45 Command output: Connection failed; retrying... (Retries left: 4)
Connection failed; retrying... (Retries left: 4)
Connection failed; retrying... (Retries left: 3)
Connection failed; retrying... (Retries left: 2)
Connection failed; retrying... (Retries left: 1)
Error: Request timed out.
CLI_VERSION: Latest - 3.41.2 available (current: 3.41.1). Please update by using
sion upgrade'

2024/04/23 16:53:45 Failed to download NGC model repo to local folder: exit st
```

Retry the model import request again.

PATCH API does not work

PATCH model/model version API are currently unsupported.

Cloudera AI Inference is unable to discover the CML model registry

In certain cases, the Cloudera AI Inference is unable to discover the CML model registry.

After upgrading CML registry, follow the *Manually Updating Model Registry Configuration* steps listed in the Troubleshooting section or delete and create model-registry to ensure that Cloudera AI Inference continues to work.

Related Information

[Debugging the model import failure](#)

[Manually Updating Model Registry Configuration](#)

Updating CML Registry configuration

After the CML Registry cluster is created, the following patch needs to be applied to the Kubernetes cluster for the standalone API authentication to work.

```
kubectl get configmap knox -n knox -o yaml | sed 's|https://consoleauth.cdp.cloudera.com/oauth2/keys|https://consoleauth.altus.cloudera.com/oauth2/keys|g' | kubectl apply -f -
```

For information on how to get the kubeconfig for the CML Registry cluster, which is similar to the CML Workspace Cluster, see *Manage remote access*.

Related Information

[Manage remote access](#)

Troubleshooting issues with Model Registry API

Learn about some of the recommended series of steps to perform when troubleshooting issues related to the Model Registry API.

Cloudera AI Inference cannot discover model registry

Learn about scenarios or issues that may be resolved by using the steps in the Manually Updating Model Registry Configuration solution mentioned below.

- ML Serving Installed Before Model Registry

If the model registry has not been created yet, when ML Serving is installed it will not be able to generate the ConfigMap.

- Model Registry Upgraded After ML Serving Installed

When the Model Registry is upgraded, it will require an updated ConfigMap.

- Model Registry Certificate Expired

After 90 days, Model Registry will have a certificate update, it will require an updated ConfigMap.

Manually Updating Model Registry Configuration

If you upgrade the CML Registry after creating your Cloudera AI Inference service cluster, the model registry configuration stored by the AI Inference service will get out of sync. Follow the steps below to manually reconcile the configuration, so that AI Inference service will be able to connect to the model registry.



Important: You are required to access multiple clusters when performing the below steps. For information configuring and switching Kubeconfig's between multiple clusters, see [Configure Access to Multiple Clusters](#)

1. Get KubeConfig for ML Serving Cluster.

- In a terminal where **CDP CLI is installed**, run the `cdp ml list-ml-serving-apps` command.
- Find the `appCrn` of the ML Serving instance you wish to update.

```
~/go/mlx-crud-app git:(master) 01:40 pm (1.452s)
cdp ml list-ml-serving-apps
2024-04-23 13:40:10,867 - MainThread - cdpcli.clidriver - WARNING - You are running an INTERNAL release of the CDP CLI, which has
different capabilities from the standard public release. Find the public release at: https://pypi.org/project/cdpcli/
{
  "apps": [
    {
      "appName": "cml-systest-aws",
      "appCrn": "crn:cdp:ml:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:mlserving:2c982e74-c60e-43b9-be85-357cfff1d419",
      "environmentCrn": "crn:cdp:environments:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:environment:3715dcbe-48a8-4292-94a0-f34e44c3bf35",
      "status": "Running"
    }
  ]
}
```

- Retrieve the KubeConfig with this command:

```
cdp ml get-ml-serving-app-kubeconfig --app-crn <YOUR-APP-CRN> |jq .kubeconfig|yq --prettyPrint
```

Here:

- `cdp ml get-ml-serving-app-kubeconfig --app-crn <YOUR-APP-CRN>`: returns raw json response containing kubeconfig.
- `|jq .kubeconfig`: returns only the actual kubeconfig (removes json wrapper).
- `|yq --prettyPrint`: prints the kubeconfig response into a format both machine and user-readable.

```
~/go/mlx-crud-app git:(master) 02:58 pm (1.387s)
cdp ml get-ml-serving-app-kubeconfig --app-crn crn:cdp:ml:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:mlserving:2c982e74-c60e-43b9-be85-357cfff1d419|jq .kubeconfig|yq --prettyPrint
2024-04-23 14:58:31,684 - MainThread - cdpcli.clidriver - WARNING - You are running an INTERNAL release of the CDP CLI, which has
different capabilities from the standard public release. Find the public release at: https://pypi.org/project/cdpcli/
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUJzQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJQXZzUm0rYys1eTB3RFFZSKtvWkLodmN0
VFfTEJRQXGdGEVUjUFR0ExVUUKQXhNS2EzZm1aWEp1WlshbG6N6QWVgdzB5TkRBME1qTkh0ak0zTkrYUz3MHpOREEwTwpFeE5qUXl0RGxhTUJVeApFekF5Qmd0VkJBT
RDBxQXW1WewJtVjBawE13Z2ZFaU1BMEddU3FHU0lIM0RRRUJBUVVBQTRJQR3QXdn20VLCkFvSUJBUURCWElpTE95TVI4L01yaGZ25khNa05ua3dJRzdXTmR5ZzhPb2
UZDBxQkh6SETMQzBsdFNpYjR3b1IKZTNDU2w0UEdlauROMFY3ZEFMZ0JVCm44ZmpIeHhJWUZpZzVoWk2cHV2ZTU5MGJNZFJXUXpYSW1TZ244cHJoZApJK3lWbEVCMkc
MVVWYUhhXaBzdQdQ2dXNTUE1QdGJ2VWJHcHRxU1JDU3l3K0hJQkJKUE0zdWhtcJFVXkhTQzZFCmh5ZWZDR0dvNjhGTCTRc2JteKR0cTFa0WV2V25lE1XeUcrd1LHQzNm
05pc3EzVWVNdU9sY0tV1hBY2lJMGgKcDA2ajlUbGVZVXxwdE50b2swVnovcUxkOGsrZENuY2ExR1FBWGHtNGNJayt5UERhcmNxmNRR1JNWEU5b21raQp6N1NjUW0hU
JrdHh0V0t1QaWRsUUFxNzQ3ZHVhQWdQdWQKfBR2pXVEJYUUE0R0ExVWREd0VCL3dRRUF3SUNWREFQCKJnTLZiUk1CQWY4RUJQUURBUUgVtUIiR0ExVWREZ1FXQkJRy0xRUJ
PMLh0ThQZkLTczRITHdSWnRWUzRUQVYkQmdOVkhSRUVEakFNZ2dWcmRXSxmxjbTVsZEdWek1BMEddU3FHU0lIM0RRRUJDD1VBQTRJQkFRQThXZmt3SXE1Zwp0TjAxd2V
azU2MFNZRjZkdQkRR2pwWlta0TJUM0JnN1RYTmVRSXYraXd50Fd1ZjZT0VpRR0tQaXhVl2Z6aXN4CnRhajk4Wgt0NnpUbEhZVEFvK3hoY1FRaDdNZVpQeUFUEYzFvVmV
GLaYkFN0TYZDkWRDZa5BkSkQ2QnFHeTIRKTMydUJGWEhPa3IzWDYrNDhlld1phYVh3b0RkaEU4QlFMVXVy0E5JeDYvMwt4MUhDSVcrMVpxT2cx0R3dTk5Ygo2M2o0T
dQaVJoR1ExRVJ1UHVob2xjYW51cXVQK2lDYTdBb29HV2haSTZPUVZ3MjJ0RVVhSTVD0GNOv3BkREpsCjd0aGFZEZzL3b1RISTc0bkdpR2xKNWt2T3JLRlJlU3MGg0VSs2dn
HTWd1Sjds1Q2TKxkSU5wZURKRTNCTHBNb24KYjBGV2dLN0xqV0w5Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
server: https://7FD9E98AABFFCB22A61AE5AF9E1815A0.gr7.us-west-2.eks.amazonaws.com
name: ml-a66eb62a-196
contexts:
- context:
  cluster: ml-a66eb62a-196
  user: ml-a66eb62a-196
  name: ml-a66eb62a-196
users:
- name: ml-a66eb62a-196
  user:
```

- Direct the output to the standard kubeconfig location:

```
cdp ml get-ml-serving-app-kubeconfig --app-crn <YOUR-APP-CRN> |jq .kubeconfig|yq --prettyPrint>~/.kube/config
```

Make sure this is the active kubeconfig by running the `kubect1 config view` command. The output of the above command should now shows the kubeconfig data returned above.

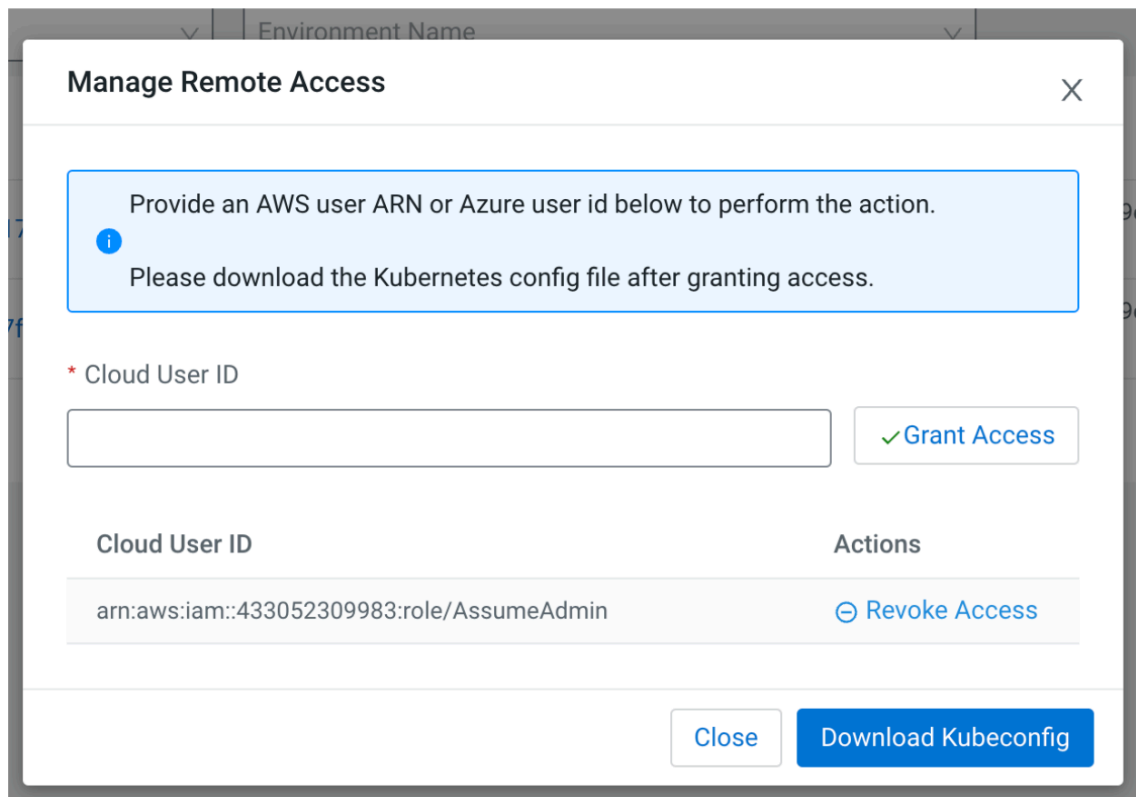
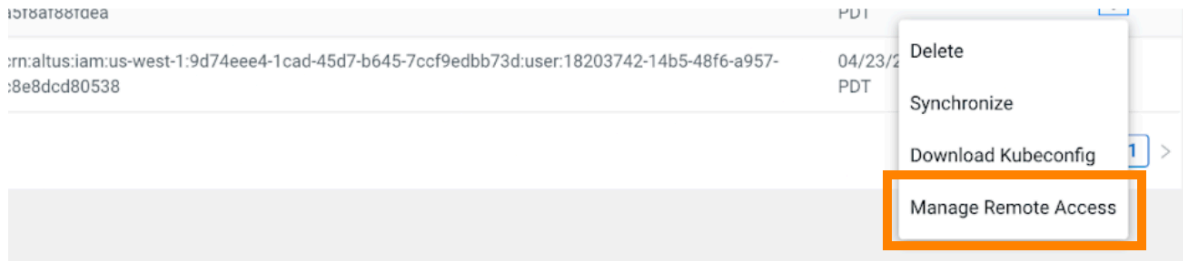
2. Grant your user ML Serving App Access

Add your cloud provider user identifier to the ML Serving App access control list. Use the following command, with your ARN, or other cloud provider user identifier.

```
cdp ml grant-ml-serving-app-access --resource-crn crn:cdp:ml:us-west-1:9
d74eee4-1cad-45d7-b645-7ccf9edbb73d:mlserving:2c982e74-c60e-43b9-be85-35
7cffb1d419 --identifier <Your Cloud Provider User ID>
```

3. Get New ConfigMap Data

- a. Find the new model registry ConfigMap. Use the grant-model-registry-access API in CDP CLI to add your user name to the new model registry, or use the UI, as shown:



- b. After your user ARN has been granted access to the Model Registry, get the ConfigMap data in the following way:
- Download the Model Registry Kubeconfig
 - Set the Model Registry Kubeconfig (can save it to ~/.kube/config)
 - `kubectl get cm -n mlx`: lists all the available ConfigMaps
 - `kubectl describe cm jwks-rootca -n mlx`: Returns the TLS Certificate for Model Registry
 - `cdp ml list-model-registries`: The response will contain the domain for the updated Model Registry
- c. Copy the entire rootca.pem output from the command

```
kubectl describe cm jwks-rootca -n mlx
```

```
Data
====
rootca.pem:
-----BEGIN CERTIFICATE-----
MIIFmDCCA4CgAwIBAgIQU9C87nMp0IFKYpfvOHFHFDANBgkqhkiG9w0BAQsFADBM
MQswCQYDVQQGEwJVUzEzMDEGA1UEChMqKFNuQUdJTkcpIEludGVybmV0IFNlY3Vy
aXR5IFJlc2VhcmNoIEdyb3VwMSIwIAAYDVQQDExoU1RBR010RYkgUHJldGVuZCBQ
ZWFiYfFgxMB4XDTE1MDYwNDExMDQzOFoXDTM1MDYwNDExMDQzOFowZjELMAkGA1UE
BhMCVVMxMzAxBgNVBAoTKihTVEFHVSU5HKSBJbnRlcm5ldCBTZWN1cml0eSBSZXNL
YXJjaCBHcm91cDEiMCAGA1UEAxMZKFNUQUdJTkcpIFByZXRLbmQgUGVhciBYMTCC
AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBALbagEdDTa1QgGBWSYkyMhsc
ZXEN0BaVRTMX1hceJENgsL0Ma49D3MilI4KS38mtkmdF6cPwnL++fgehT0FbRHZg
j0Er8UAN4jh6omjrbTD++VZneTsMVaGamQmDdFl5g1gYaigkkmx80iC068a4QXg4
wSyn6idipKP8utsE+x1E28SA75H0Yqpdrk4HGxuULvlr03wZGTIf/oRt2/c+dYmD
oaJhge+G0rLAEQBy07+8+vz0wpNAPEx6LW+crEEZ7eBXih6VP19sTGy3yfQK5tPt
TdXXC0QMKA+pGcj/VByhmIr+0iNDC540gtvV303WpcbwknkLYC0ft2cYUyHtkst0
fRcR0+K2cZozoSwVPyB8/J9RpcRK3jgnX9lujfWA/pAbP0J2UPQFxmWFRqnFjaq6
rkqbNEBgLy+kFL1NEsRbvFbKrRi5bYy2lnms2NJJPzdNQBT/2dBZKmJqxHkxCuOQ
FjhJQNe0+Njm1Z1iATS/3rts2yZlqXksxQUzN6vnNbD8KnXRMEeOXUYvbV4lqfCf8
mS14WEbSiMy87GB5S9ucSV1XurlTG5UGcMSZOBCeUpisRPemQWUOTWIodQ5FOia/
GI+Ki523r2ruEmbmG37EBSBXdxIdndqrjy+QVAmebyDx9eVEGOIpn26bw5LKeru
mJxa/CFBaKi4bRvmdJRLAqMBAAGjQjBAMA4GA1UdDwEB/wQEAWIBBjAPBgNVHRMB
Af8EBTADAQH/MBOGA1UdDgQWBBS182Xy/rAKkh/7PH3zRKCYSyXDFDANBgkqhkiG
9w0BAQsFAAOCAgEAncDZNytDbrrVe68UT6py1lfF2h6Tm2p8ro42i87WwyP2LK8Y
nLHC0hvNfWeWmjZYQBQfGC5c7aQRezaktHLdmrNKHkn5kn+9E9LCjCaEsyIIIn2j
qdHlAkepu/C3KnNtVx5tW07e5bvIjJSckwCDbP3akWQixPpRFAsnP+ULx7k0a01x
qAeaAhQ2rgo1F58hcflgqKTXnpPM02intVfiVVkX5GXpJjk5EOQtLceyG0rkxLM/
sTPq4UrnyPmsqSagWV3HcuLYtDinc+nukFk6ER4XkzXBbwKajl0YjztfrCIH0n5Q
CJL6TERVDdbM/aPlv8kJ1sWGLuvvWYZMYqLzDuL//rUF10qEMWaXVZV51KoS9DY/
```

The domain of the new model registry is contained in the list-model-registries response:

```

~ 05:43 pm (1.773s)
cdp ml list-model-registries
2024-04-23 17:43:36,327 - MainThread - cdpcli.clidriver - WARNING - You are running an INTERNAL release of the CDP CLI, which has
different capabilities from the standard public release. Find the public release at: https://pypl.org/project/cdpcli/
{
  "modelRegistries": [
    {
      "id": 990,
      "creator": "crn:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:user:1f77129b-9cb0-4a23-970f-a5f8af88fdea",
      "status": "installation:finished",
      "environmentCrn": "crn:cdp:environments:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:environment:3715dcbe-48a8-4292
-94a0-f34e44c3bf35",
      "createdAt": "2024-04-23T22:25:58.569000+00:00",
      "crn": "crn:cdp:ml:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:model_registry:dfcff53f-4f0f-4a54-9eaf-4f1cd384387a"
    },
    {
      "environmentName": "eng-ml-dev-env-aws",
      "workspaceName": "model-registry-ml-17bf05df-050",
      "machineUserCrn": "crn:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:machineUser:cml_env_machine_user_3715
dcbe-48a8-4292-94a0-f34e44c3bf35/5ce78dc4-e84d-498d-a2a0-ebef1d0dae2d",
      "serviceName": "model-registry-ml-17bf05df-050",
      "domain": "https://modelregistry.ml-17bf05df-050.eng-ml-d.xcu2-8y8x.dev.cldr.work"
    }
  ]
}

```


4. Apply ConfigMap Update

- a. Update the KUBECONFIG back to the ML Serving kubeconfig.
- b. Edit the ConfigMap of the ML Serving Cluster:
 - `kubectl edit cm modelregistry-config-controlplane -n serving`: Update `tls.crt` with the data from above.
 - `kubectl describe cm api-config -n serving`: Update `model.registry.url` with the data from above
- c. Restart the deployment to force the cm changes to take effect
 - `kubectl scale deployment api -n serving --replicas=0`
 - `kubectl scale deployment api -n serving --replicas=1`

Debugging the model import failure


To debug errors that occurred on the Model Registry server, you can access the logs found in the API v2 pod.

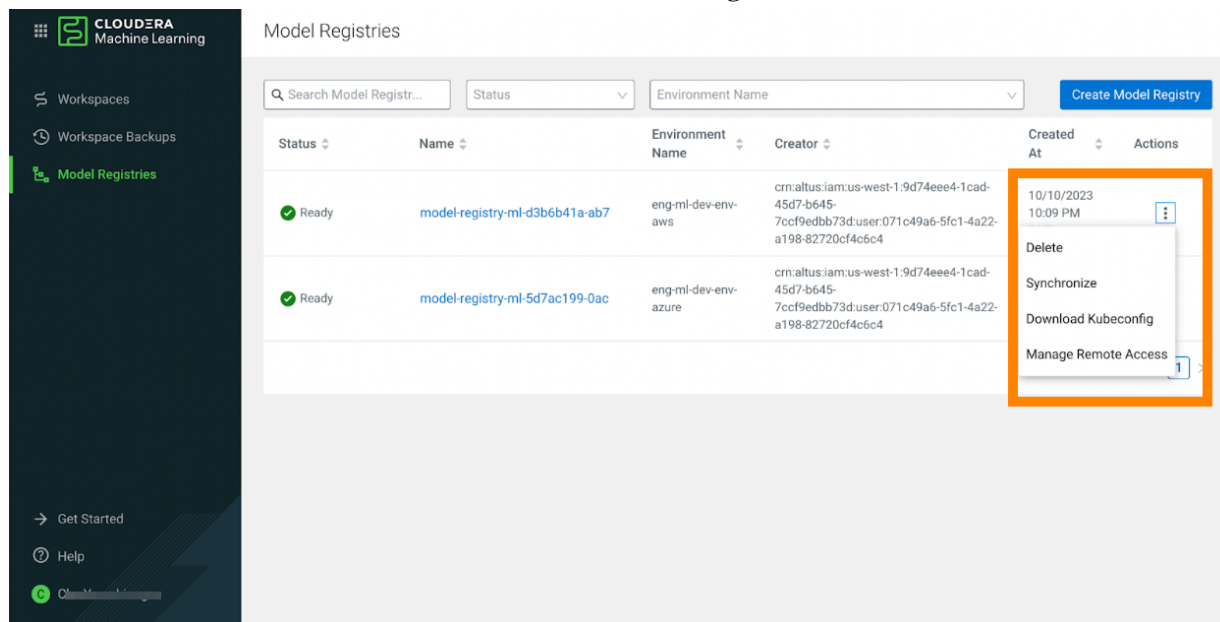
About this task

Accessing logs from the Model Registry Kubernetes cluster

You can obtain the kubeconfig for the model registry cluster.

1. In the **Cloudera Data Platform (CDP)** console, click the **Machine Learning** tile.
2. Click **Model Registries** in the left navigation menu. The **Model Registries** page displays.
- 3.

In the **Actions** menu, click  and select **Download Kubeconfig**.



The screenshot shows the Cloudera Machine Learning console. On the left, the 'Model Registries' section is selected in the navigation menu. The main panel displays a table of Model Registries. The first entry, 'model-registry-ml-d3b6b41a-ab7', is highlighted. An actions menu is open for this entry, showing options: 'Delete', 'Synchronize', 'Download Kubeconfig', and 'Manage Remote Access'. The 'Download Kubeconfig' option is highlighted with an orange box.

Status	Name	Environment Name	Creator	Created At	Actions
Ready	model-registry-ml-d3b6b41a-ab7	eng-ml-dev-env-aws	cm:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:user:071c49a6-5fc1-4a22-a198-82720cf4c6c4	10/10/2023 10:09 PM	<ul style="list-style-type: none"> Delete Synchronize Download Kubeconfig Manage Remote Access
Ready	model-registry-ml-5d7ac199-0ac	eng-ml-dev-env-azure	cm:altus:iam:us-west-1:9d74eee4-1cad-45d7-b645-7ccf9edbb73d:user:071c49a6-5fc1-4a22-a198-82720cf4c6c4		

In AWS, you need to add your identity under Manage Remote Access to access the Kubernetes cluster.

You must add your identity under Manage Remote Access. For information on granting remote access, see *Granting Remote Access to ML Workspaces*. After the kubeconfig is set up, run the following `kubectl` command to get logs for the model registry pod:

```
kubectl logs <model registry pod name> -n mlx
```

Related Information

[Granting Remote Access to ML Workspaces](#)

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`

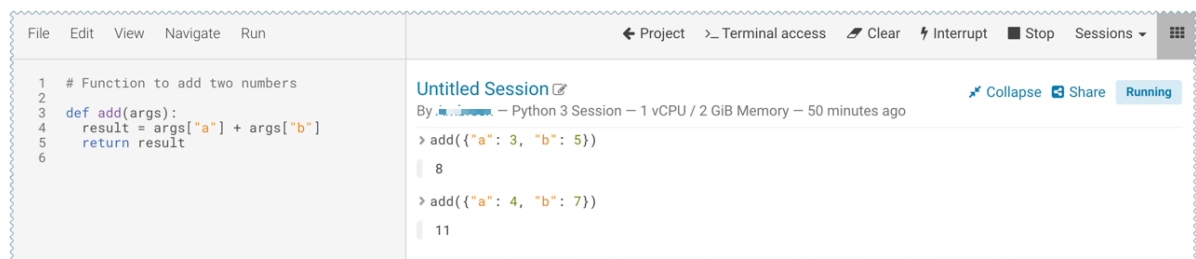
```
def add(args):  
    result = args["a"] + args["b"]  
    return result
```



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.

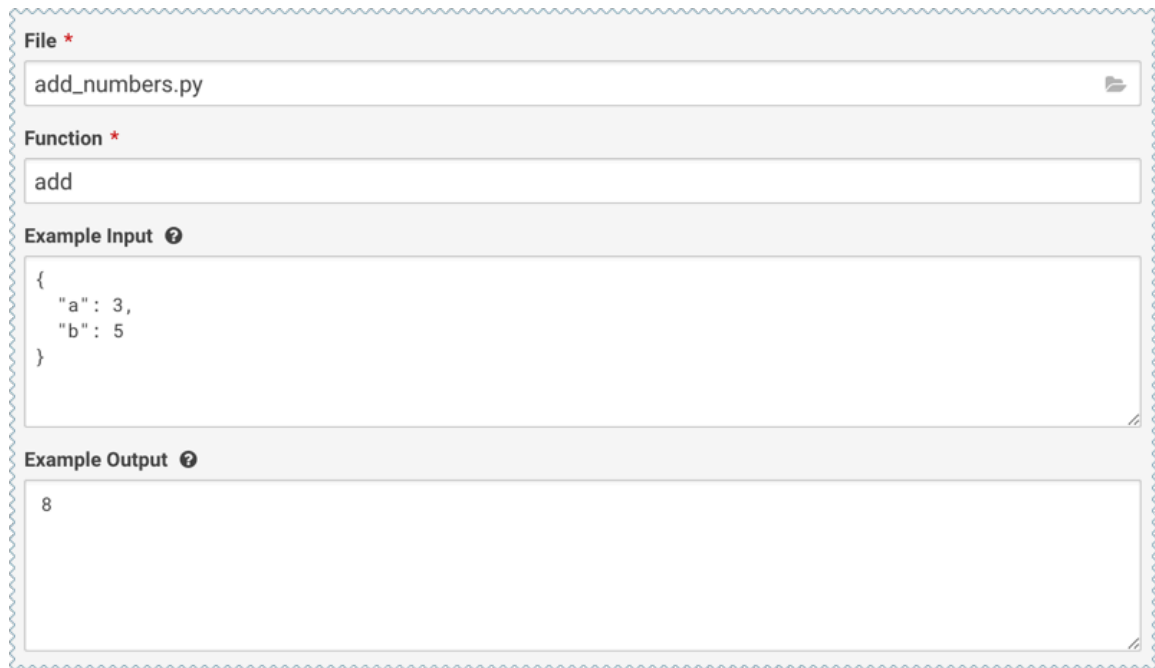
4. Before deploying the model, test it by running the `add_numbers.py` script, and then calling the `add` function directly from the interactive workbench session. For example:

```
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. In **Deploy Model as**, if the model is to be deployed in a service account, select **Service Account** and choose the account from the dropdown menu.
- e. 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`



The screenshot shows a web form for creating a new model. It contains the following fields and values:

- File ***: `add_numbers.py`
- Function ***: `add`
- Example Input ?**:

```
{  
  "a": 3,  
  "b": 5  
}
```
- Example Output ?**: `8`

- f. Select the resources needed to run this model, including any replicas for load balancing. To specify the maximum number of replicas in a model deployment, go to **Site Administration Settings Model Deployment Settings**. The default is 9 replicas, and up to 199 can be set.



Note: The list of options here is specific to the default engine you have specified in your Project Settings: **ML Runtimes** or **Legacy Engines**. Engines allow kernel selection, while **ML Runtimes** allow Editor, Kernel, Variant, and Version selection. Resource Profile list is applicable for both **ML Runtimes** and **Legacy Engines**.

- g. Click **Deploy Model**.

- 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.

Add Two Numbers Building Stop Deploy New Build

Overview Deployments **Builds** Monitoring Settings

Build	Status	File	Function	Kernel	Engine Image	Created By	Created At	Comment	Actions
1	Building	add_numbers.py	add	python3	Base Image v	ambreen	Jun 5, 2018, 5:44 PM	Initial revision.	Delete

```

Sending build context to Docker daemon 15.05 MB

Step 1/16 : FROM docker.repository.cloudera.com/cdsw/engine:
----> f8955770daa1
Step 2/16 : ENTRYPOINT node /app/model-runtime/model-server.js
----> Running in 58038f1e58d5

```

- 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:

- Create a project with the Python template and a legacy engine.
- Start a session.
- Run `!pip3 install sklearn`
- 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 in the public cloud are highly available subject to the following limitations:

- Model high availability is dependent on the high availability of the cloud provider's Kubernetes service. Please refer to your chosen cloud provider for precise SLAs.
- Model high availability is dependent on the high availability of the cloud provider's load balancer service. Please refer to your chosen cloud provider for precise SLAs.
- In the event that the Kubernetes pod running the model proxy service 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

[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 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 checkpoint 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 in the public cloud are highly available subject to the following limitations:
 - Model high availability is dependent on the high availability of the cloud provider's Kubernetes service. Please refer to your chosen cloud provider for precise SLAs.
 - Model high availability is dependent on the high availability of the cloud provider's load balancer service. Please refer to your chosen cloud provider for precise SLAs.
 - In the event that the Kubernetes pod running the model proxy service 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

[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=<value> -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.

Test Model

Input

```
{
  "a": 3,
  "b": 5
}
```

Test **Reset**

Result

Status	● success
Response	8
Replica ID	add-two-numbers-1-1-86b9b58b7b-g6s8r

- 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.

Add Two Numbers

Overview [Deployments](#) [Builds](#) [Monitoring](#) [Settings](#)

Description

Sample Code

Add two numbers.

Shell Python R

```
curl -H "Content-Type: application/json" -H "Authorization: Bearer
<place API key here>" -X POST https://
/model -d '{"accessKey": "mgc4w3rdi4
3x28fy4h8e8:", "request": {"a": 3, "b": 5}}'
```


Securing Models

You can secure your Cloudera Machine Learning models using Access keys or API keys.



Important: Cloudera Data Platform (CDP) Public Cloud allows customers to maintain full ownership and control of their data and workloads and is designed to operate in some of the most restricted public cloud environments. Since CDP Public Cloud runs in a customer's cloud account, Security and Compliance is a shared responsibility between Cloudera and its public cloud customers. For more information, see *Cloudera's Shared Responsibility Model*.

Related Information

[Cloudera's Shared Responsibility Model](#)

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.

Add Two Numbers

Overview Deployments Builds Monitoring **Settings**

Name
Add Two Numbers

Description
This model takes two numbers as input and returns their sum.

Access Key [Regenerate](#)

mgc4w3r di43x28fy4h8e8swsda4jyfoq **← Access Key is required to make requests on this model**



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:

Add Two Numbers

Overview Deployments Builds Monitoring Settings

Id	Build	Status	Deployed At	Stopped At	Deployed By
4	3	Deployed	Oct 20, 2021, 03:34 PM		csso_wclmens
3	2	Stopped	Oct 20, 2021, 03:16 PM	Oct 20, 2021, 03:33 PM	csso_wclmens

Model

Id 2

Name Add Two Numbers

Description Add two numbers.

Re-deploy This Build

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.

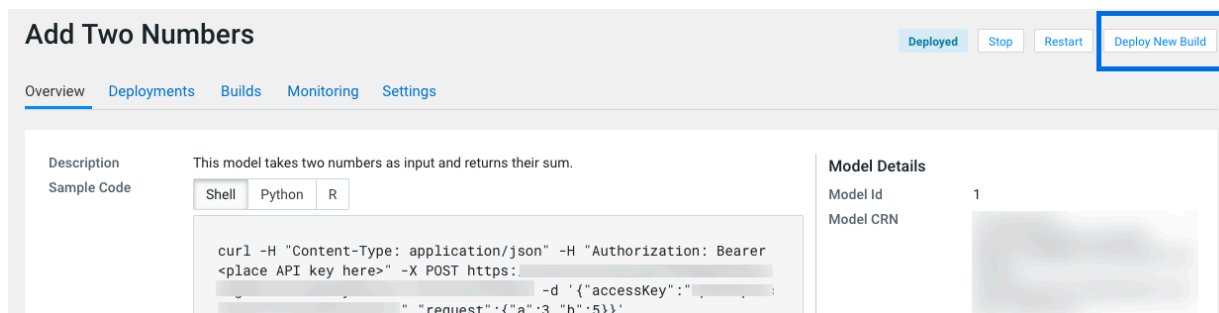


Warning: Currently, Cloudera Machine Learning does not allow you to create a new build for a model without also deploying it. This combined with the fact that you can only have one active deployment per model means that once the new model is built, the current active deployment will go offline so that 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

[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.

Using Cloudera Copilot

Learn how to configure and use Cloudera Copilot with Cloudera AI Inference service and Amazon Bedrock models.

Cloudera Copilot is an AI-powered coding assistant designed for seamless integration within JupyterLab ML Runtimes. With its chat interface and comprehensive code completion features, Cloudera Copilot enhances the development experience for machine learning projects. It offers compatibility with model endpoints deployed in Cloudera AI Inference service as well as Amazon Bedrock models, providing developers with flexibility and efficiency in their workflows.

Cloudera AI Inference service vs Amazon Bedrock

Cloudera AI Inference service model endpoints may be a good choice if you are concerned about proprietary data being sent to a third-party service provider. With Cloudera AI Inference service, you can run your own models and ensure that your proprietary data stays within your cloud deployment.

If you are not concerned about proprietary data being sent to a third-party service provider, and you do not expect high volumes of Cloudera Copilot usage, then using Amazon Bedrock models may be a simpler and more cost-effective solution.

Configuring Cloudera AI Inference service and Amazon Bedrock to set up Cloudera Copilot

To use Cloudera Copilot, as a Site Administrator, set up Cloudera AI Inference service model endpoints or configure the credentials in Amazon Bedrock depending on where you want to deploy your custom model.

For Cloudera AI Inference

Cloudera AI Inference service is a production-grade serving environment for traditional, generative AI, and LLM models. It is designed to handle the challenges of production deployments, such as high availability, fault tolerance, and scalability. Follow the steps to configure Cloudera AI Inference service model endpoints to use Cloudera Copilot.

1. [Configure authentication, and authorization, and import a model from NGC.](#)
2. [Create a Cloudera AI Inference service instance.](#)
3. [Create a model endpoint using UI.](#)

No additional credentials need to be configured to use Cloudera AI Inference service model endpoints with Cloudera Copilot.

For Amazon Bedrock

To use Cloudera Copilot, as a Site Administrator, you can configure the credentials to use Amazon Bedrock models.

1. Generate a pair of Access and Secret keys through AWS IAM. For more information, see [Manage access keys for IAM users](#).
2. In the **Cloudera Data Platform** (CDP) console, click the **Machine Learning** tile.
The **Machine Learning Workspaces** page displays.
3. Click on the workspace name.
The **Workspaces Home** page displays.
4. Click **Site Administration** in the left navigation menu.
The Site Administration page displays.
5. Click **Settings Environment Variables** and add the following obtained in Step 1:
 - AWS_SECRET_ACCESS_KEY
 - AWS_ACCESS_KEY_ID
 - AWS_DEFAULT_REGION

Configuring Cloudera Copilot

After setting up credentials, you must make configuration changes in the Cloudera Machine Learning UI before using Cloudera Copilot.

Choosing a model

Models vary in accuracy, and cost. Larger models will provide more accurate responses but will cost more. For Cloudera AI Inference service models, larger models require more expensive GPU hardware to run on, while in Amazon Bedrock, larger models will cost more per prompt.

Language models vs Embedding models

Cloudera Copilot supports the following model types:

- Language models: These are used for code completion, debugging, and chat.
- Embedding models: These are used for Retrieval Augmented Generation (RAG) use cases. This allows you to augment language model responses with specific information that a language model is not aware of. For example, you can provide internal company documents that map company acronyms to their definitions.

Recommended models

- Language models:
 - Llama 3.1 Instruct 70b (AI Inference)
 - Claude v3 Sonnet (Amazon Bedrock)
- Embedding models:
 - E5 Embedding v5 (AI Inference)
 - Titan Embed Text v2 (Amazon Bedrock)

Procedure

1. In the **Cloudera Data Platform** console, click the **Cloudera Machine Learning** tile.
The **Cloudera Machine Learning Workspaces** page displays.

2. Click on the workspace name.
The **Workspaces Home** page displays.
3. Click **Site Administration** in the left navigation menu.
The **Site Administration** page displays.
4. Click **Settings**, and select the **Enable Cloudera Copilot** checkbox under **Feature Flags**.
A new navigation tab **Cloudera Copilot** appears at the top of the **Site Administration** page.
5. Click the **Cloudera Copilot** tab.
The **Cloudera Copilot** page displays.
6. Click the **Add Model**

Add a model to Cloudera Copilot



* Model Provider

Amazon Bedrock



* Model

anthropic.claude-v2

Set as default



The default model will be the default option when users use Cloudera Copilot. The first model added is required to be default.

Cancel

+ Add

button.

7. Select a model provider from the Model Provider dropdown list.

8. In the Model field, provide the model name:

- For Bedrock models: Select a model name from the Model dropdown list.
- For Cloudera AI Inference service models, provide the model endpoint and the `model_id` as the model name string. You can get the model endpoint and `model_id` information from the [Model Endpoint details](#) page.
 - Example Model Endpoint: `https://caii-prod-long-running.eng-ml-l.vnu8-sqze.yourcompany.site/namespaces/serving-default/endpoints/llama-31-8b-instruct-2x10g/v1/chat/completions`
 - Example model ID: `phwq-gqmd-4kos-perd`
- For Cloudera AI Inference service embedding models, provide the string of the embedding model listed in the below table.

Embedding Model Name	Model String
Mistral Embedding V2	nvidia/nv-embedqa-mistral-7b-v2
Snowflake Arctic Embed Large Embedding	snowflake/arctic-embed-l
E5 Embedding v5	nvidia/nv-embedqa-e5-v5

- Example embedding model endpoint: `https://caii-prod-long-running.eng-ml-l.vnu8-sqze.yourcompany.site/namespaces/serving-default/endpoints/mistral-7b-embedding-onnx/v1/embeddings`

The model that you add for the first time is selected as the default language model automatically and the deselect option is disabled. This is to enforce that there is always one default language model. When you add more models, you can choose any of them to be the default language model.



Note: You must ensure to select a language model for the default model and not an embedding model. The default language model is used when using the Cloudera Copilot Runtime.

9. Click Add.

Results

The model appears under Cloudera AI Inference service Models or Third Party Models depending on the model provider type you selected.

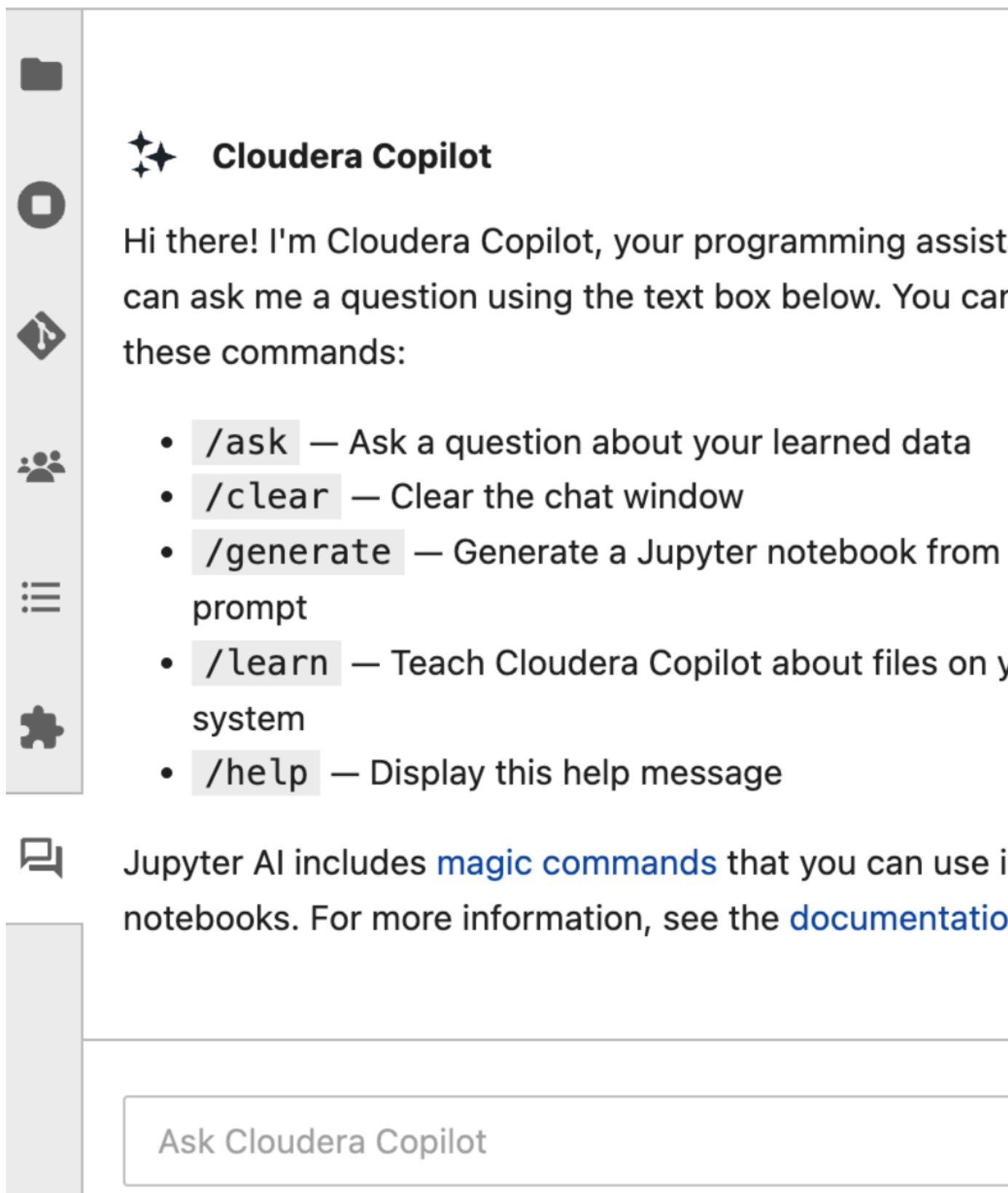
Using Cloudera Copilot


You can use Cloudera Copilot to assist you in code generation and code completion thus providing flexibility and efficiency in the Machine Learning workflows.

Procedure

1. Start a session within your project and select JupyterLab as your editor.


2. Click the  Jupyter AI Chat icon in the left navigation pane.



 **Cloudera Copilot**

Hi there! I'm Cloudera Copilot, your programming assistant. You can ask me a question using the text box below. You can use these commands:

- `/ask` — Ask a question about your learned data
- `/clear` — Clear the chat window
- `/generate` — Generate a Jupyter notebook from a prompt
- `/learn` — Teach Cloudera Copilot about files on your system
- `/help` — Display this help message


 Jupyter AI includes [magic commands](#) that you can use in your notebooks. For more information, see the [documentation](#).

Ask Cloudera Copilot

3. Start typing in the prompt and the Cloudera Copilot will assist you.

For example, enter Help me write a fibonacci function.



Important: Embedding models are not selected by default in the Copilot UI. You can click the  icon from the **Cloudera Copilot** chat interface and explicitly select the embedding model.

Example use cases for Cloudera Copilot

- Debugging/fixing code
- Code completion
- Explaining code or explaining errors
- Code generation
- Refer to [Jupyter AI documentation](#) for instructions on `/ask` `/learn` `/fix` and `/generate` commands

Magic commands

- In addition to using the chat interface, you can also use Cloudera Copilot by calling Magic Commands within your notebook. Magic commands are special lines of code that will make calls to a language model that you have configured.
- Example syntax:
 - `%load_ext jupyter_ai_magics`
 - `%%ai anthropic.claude-3-sonnet-20240229-v1:0`
 - `def fibonacci(`
- Finding model ID strings for magic commands .
 - The Amazon Bedrock models recommended for Cloudera Copilot are:
 - `anthropic.claude-3-sonnet-20240229-v1:0`
 - `anthropic.claude-3-opus-20240229-v1:0`
 - `mistral.mixtral-8x7b-instruct-v0:1`
 - `meta.llama3-70b-instruct-v1:0`
 - `amazon.titan-embed-text-v2:0`
 - For Cloudera AI Inference, the model ID string will be of the form:
 - `cloudera:<model_name>` . Here, the `model_name` is the same model ID from the Model Endpoints details page that you added when you configured Cloudera Copilot.

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:

Create a New Project

Project Name

Iris Project

Project Visibility

☐ **Private** - Only added collaborators can view the project.

☒ **Public** - All authenticated users can view this project.

Initial Setup

Blank Template Local Git

Python

Templates include example code to help you get started.

Create Project

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.

Train the Model

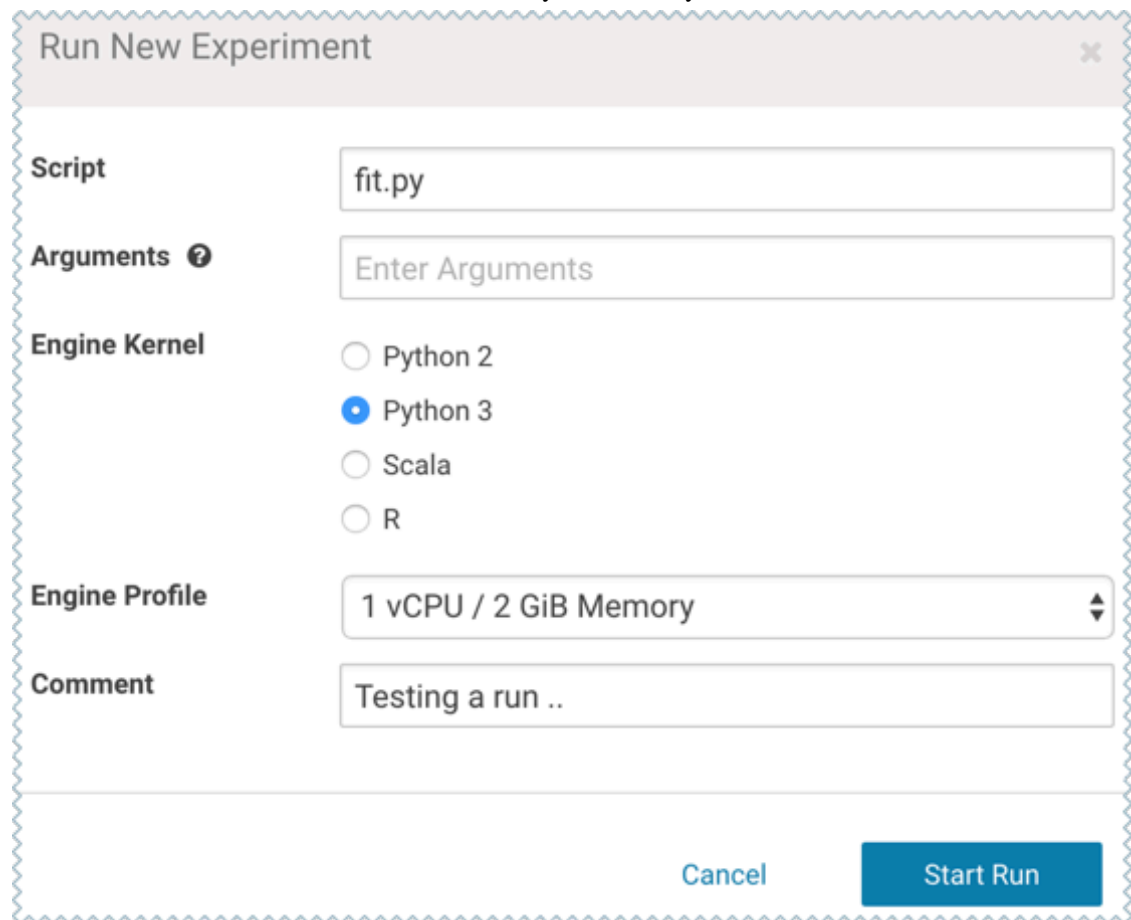
This topic shows you how to run experiments and develop a model using the `fit.py` file.

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.



The image shows a 'Run New Experiment' dialog box with a light gray header and a close button (X) in the top right corner. The dialog contains several input fields and radio buttons. The 'Script' field is a text box containing 'fit.py'. The 'Arguments' field is a text box with a question mark icon and the placeholder text 'Enter Arguments'. The 'Engine Kernel' section has four radio buttons: 'Python 2', 'Python 3' (which is selected with a blue dot), 'Scala', and 'R'. The 'Engine Profile' field is a dropdown menu showing '1 vCPU / 2 GiB Memory'. The 'Comment' field is a text box containing 'Testing a run ..'. At the bottom right of the dialog are two buttons: 'Cancel' in blue text and 'Start Run' in white text on a blue background.

Field	Value
Script	fit.py
Arguments	Enter Arguments
Engine Kernel	Python 3
Engine Profile	1 vCPU / 2 GiB Memory
Comment	Testing a run ..

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.

Run-21

Overview Session Build

Configuration

Script	fit.py
Arguments	
Comment	
Build Snapshot	cd61c8ac443de924189a55c5562d29268bbcb539
Created At	6/21/18 6:06 PM
Submitter	admin

Metrics

mean_sq_err	0.007866659505691643
r2	0.04934628330010382

Output

☐ model.pkl

Add to Project

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.

Test Model

Input

```
{  
  "petal_length": 5.4  
}
```

Test

Reset

Result

Status	● success
Response	1.8826221434150965
Replica ID	predict-petal-width-2-9-7cf557b957-5wjld