Cloudera AI

Models

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

https://docs.cloudera.com/

Legal Notice

Contents

Model Training and Deployment Overview

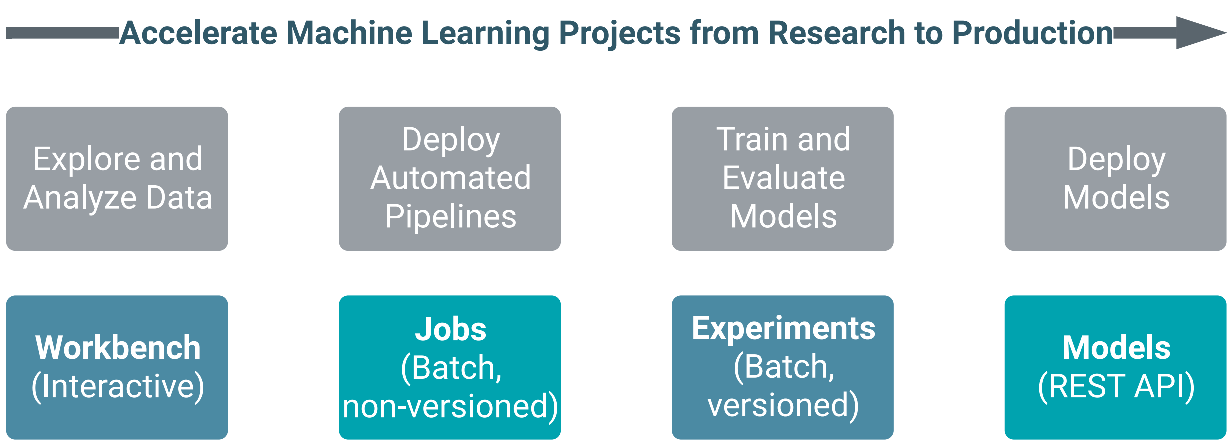
This section provides an overview of model training and deployment using Cloudera AI.

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 AI, you can deploy the complete lifecycle of a machine learning project from research to deployment.



Models

Cloudera AI 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 AI 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 AI will:

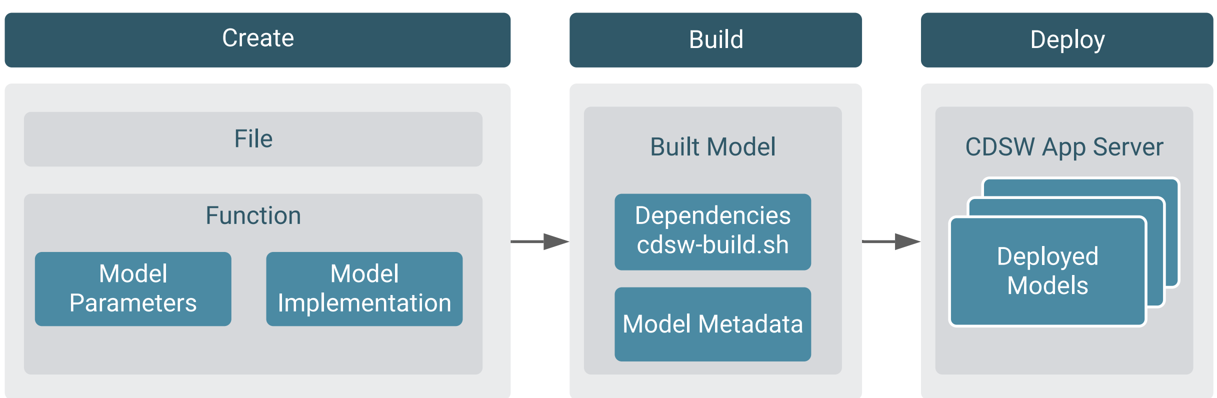
- Create a snapshot of model code, model parameters, and dependencies.
- Package a trained model into an immutable artifact and provide basic serving code.
- Add a REST endpoint that automatically accepts input parameters matching the function, and that returns a data structure that matches the function's 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 shall note that 'model' does not always refer to a specific artifact. More precise terms (as defined later in this section) shall 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 AI.

  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 AI 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 AI 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 AI. 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 AI 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 are not 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

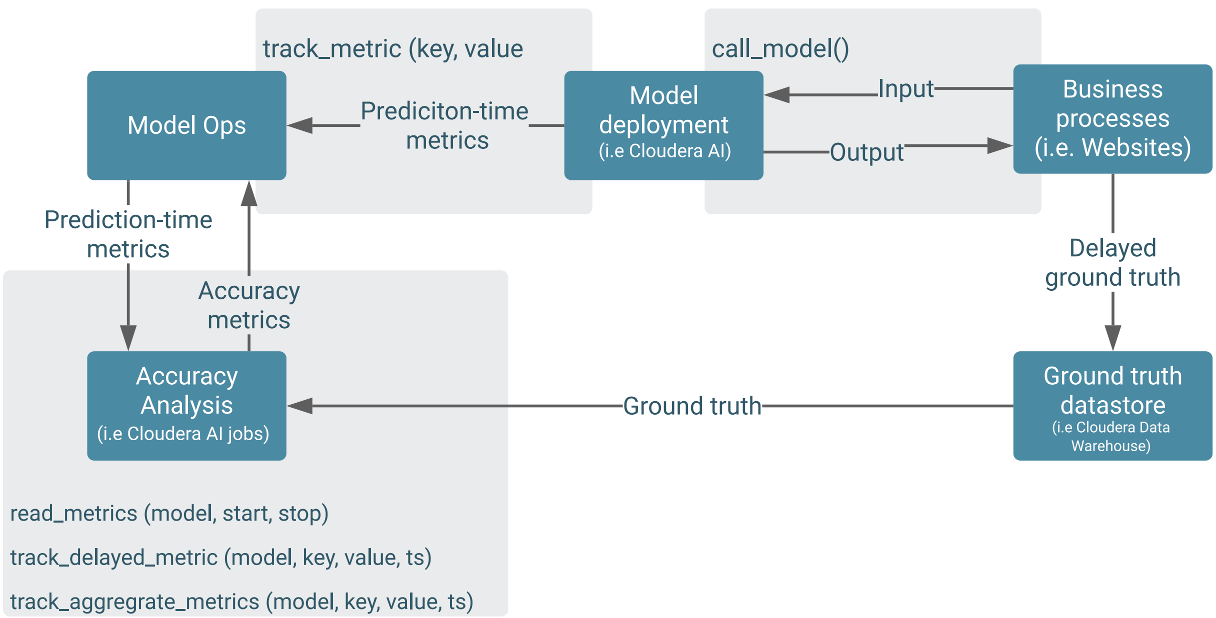
Usecase 1: Tracking accuracy of a model over time

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 AI. For each prediction thus obtained, the web application records the UUID into a datastore alongside the customer ID. The prediction itself is tracked in Cloudera AI 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 Cloudera AI 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

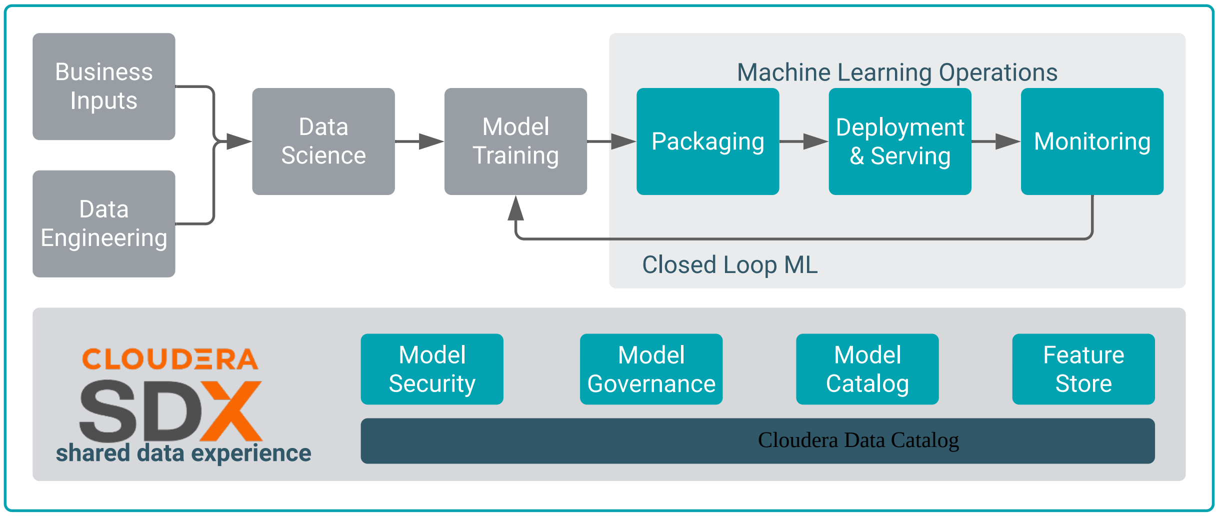
Models are often seen as a black box: data goes in, something happens, and a prediction comes out. This lack of transparency is challenging on a number of levels and is often represented in loosely related terms explainability, interpretability, and reproducibility.

- 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, Cloudera AI provides an end-to-end model governance and monitoring workflow that gives organizations increased visibility into their machine learning workflows and aims to eliminate the blackbox nature of most machine learning models.

The following image shows the end-to-end producltion ML workflow:

Figure 1: Production ML Workflow



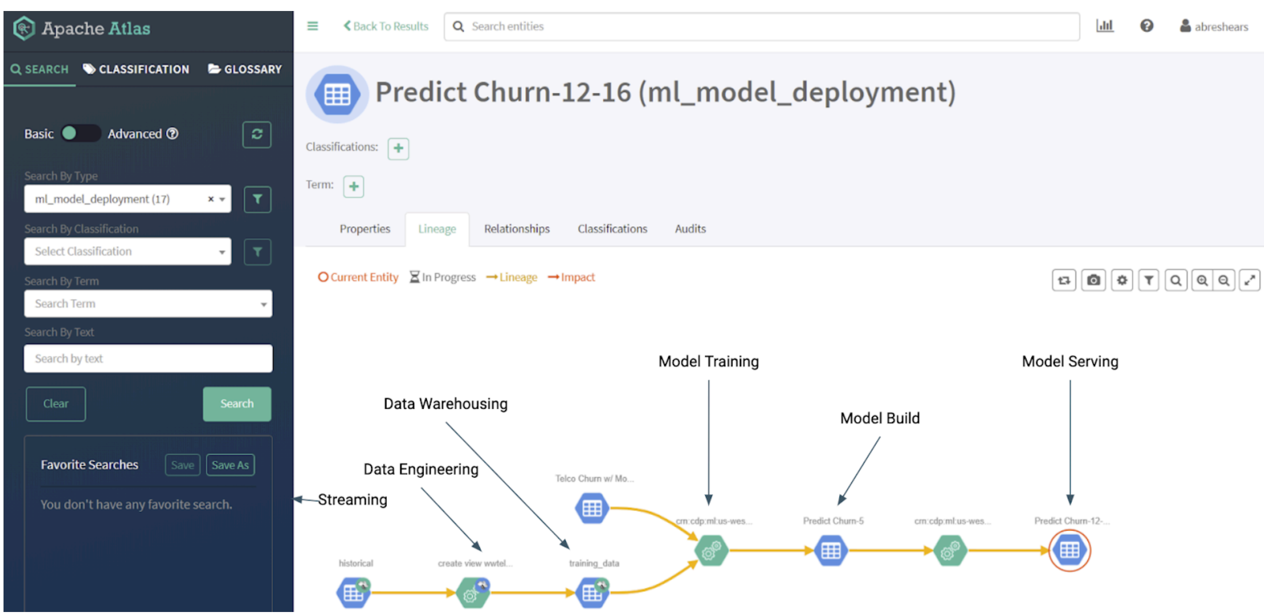
Model governance using Apache Atlas

To address governance challenges, Cloudera AI uses Apache Atlas to automatically collect and visualize lineage information for data used in Cloudera AI 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).



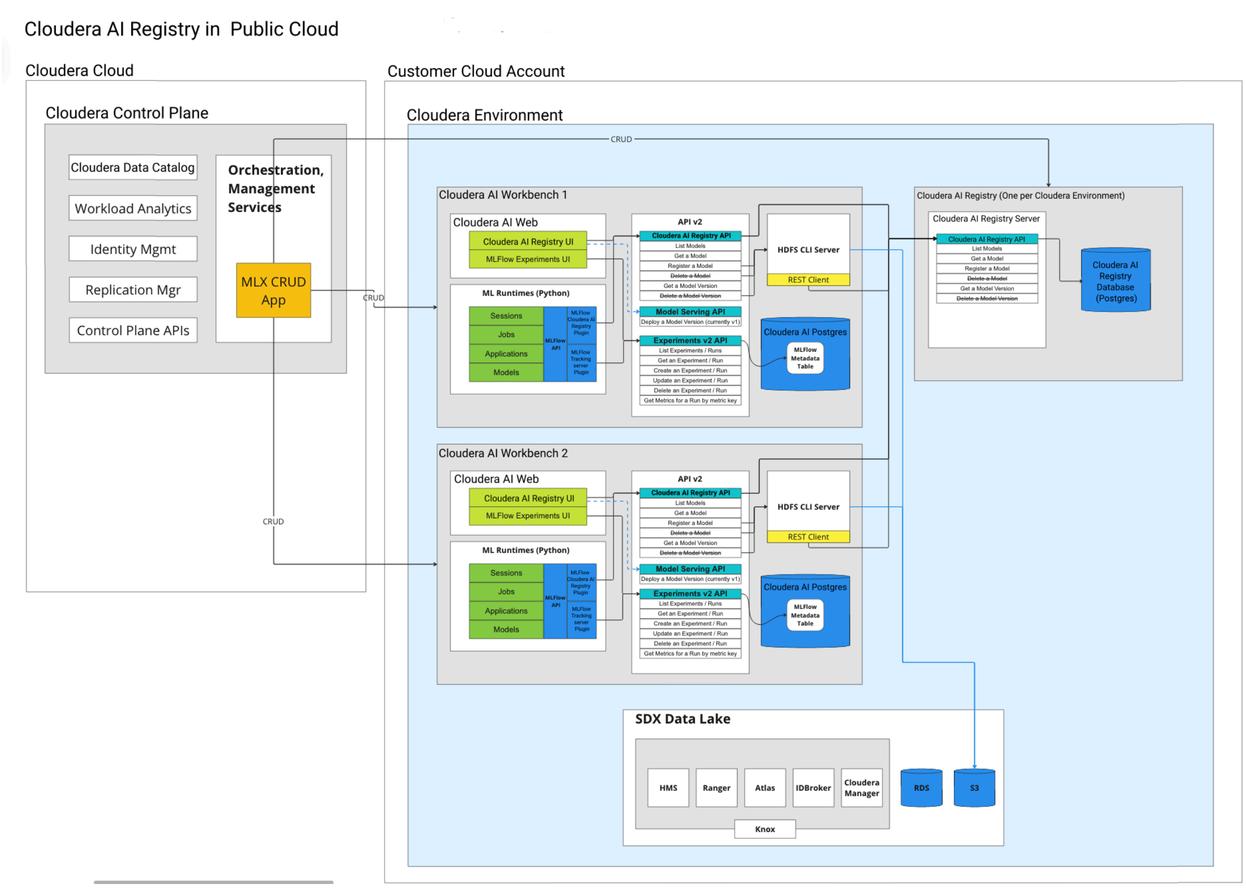
The ML metadata definition closely follows the actual machine learning workflow. Training data sets are the starting point for a model lineage flow. These data sets can be tables from a data warehouse or an embedded csv file. Once a data set has been identified, the lineage follows into training, building and deploying the model.

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

Using Cloudera AI Registry

Cloudera AI Registry is the core enabler for MLOps, or DevOps for machine learning.

Cloudera AI 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.



Cloudera AI 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.

Cloudera AI Registry instances 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 Cloudera AI Registry integrates MLFlow and maintains compatibility with the open source ecosystem.

Limitations

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

Registering and deploying models with Cloudera AI Registry

After you have set up Cloudera AI Registry, you can create, register, and deploy models with **AI 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 AI Registry user interface

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

Using the AI 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 Cloudera AI Registry with a version tag. The first time you register a model, **AI Registry** automatically creates a model repository with the first version of the model.

Before you begin

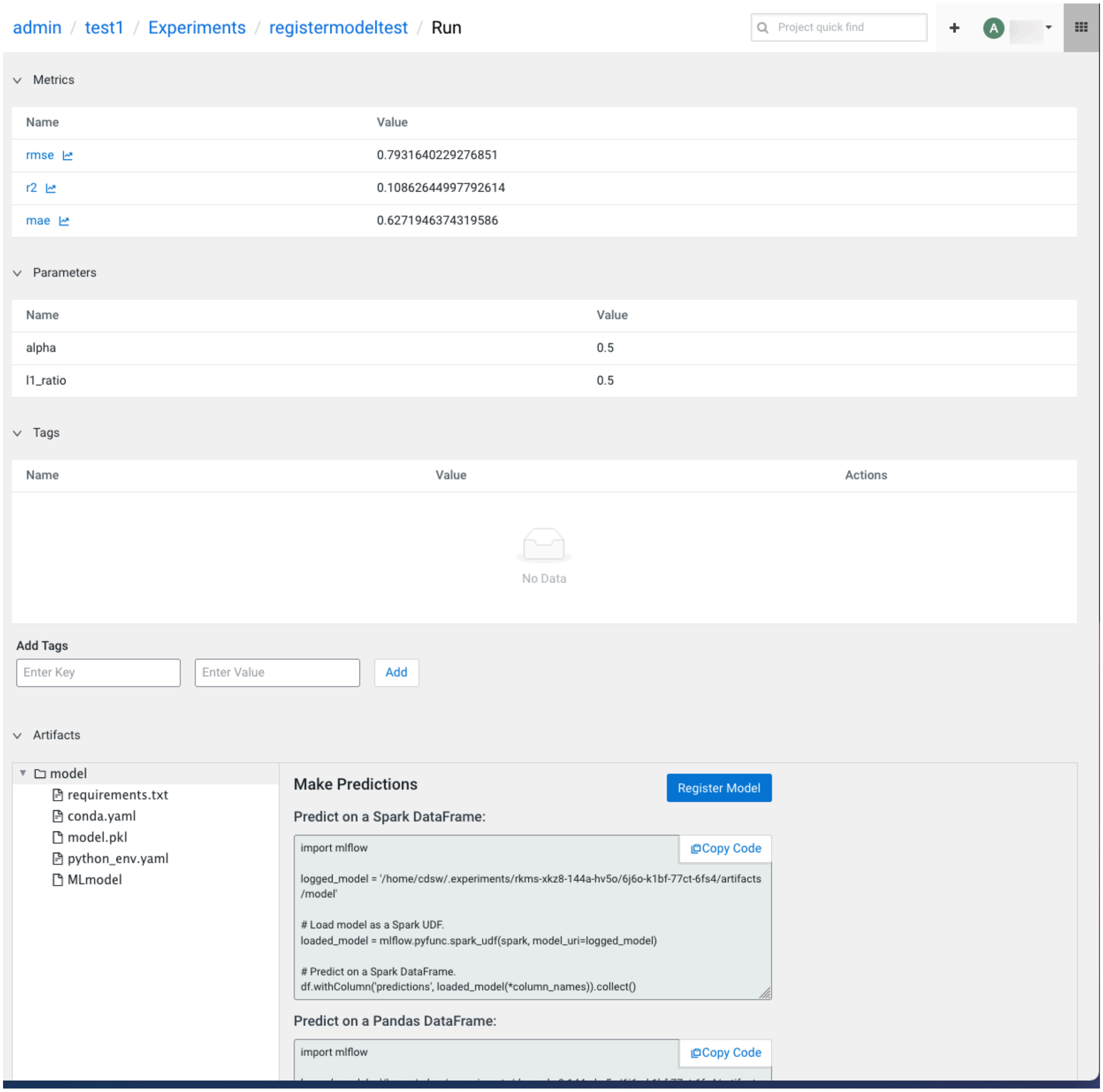
You must have permission to access a project in which the model is created before you can register the model.

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.

   AI Registry 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.

   Cloudera AI displays the Experiment Run Details page.



5. Select the run that contains the model you want to register.
6. Select Register Model to begin the registration process.

   AI 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 Cloudera AI Registry with a version tag. The first time you register a model, Cloudera AI 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_na me.

Using MLflow SDK to register customized models

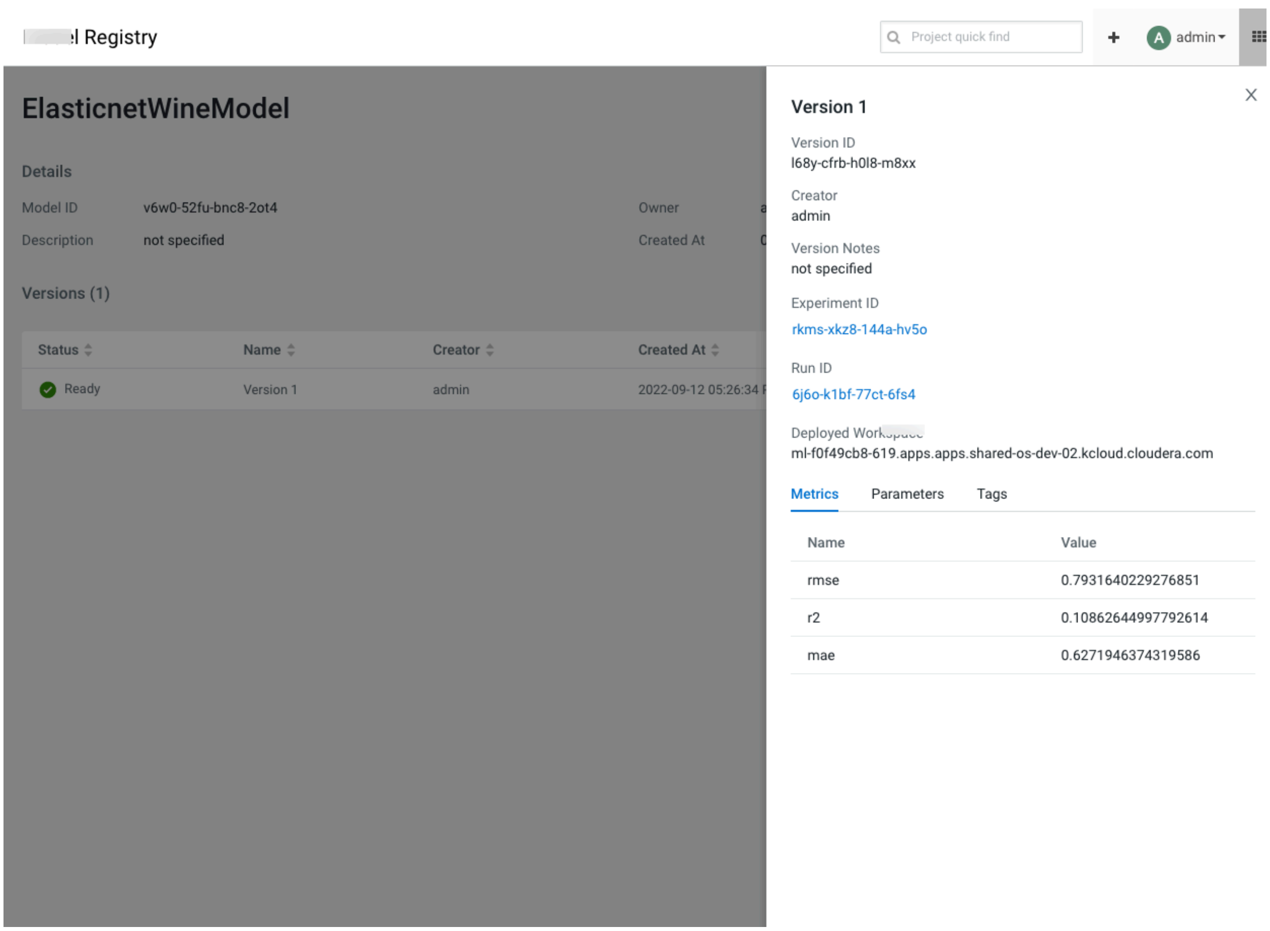
In MLflow, you can also deploy models that are not directly supported by MLFlow.

To learn more, see Serving LLMs with MLflow: Leveraging Custom PyFunc.

Viewing registered model information

1. From the Projects page in Cloudera AI, select **AI Registry** from the navigation pane. On the main **AI 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.

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



Creating a new version of a registered model

Follow the instructions to 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 AI Registry page

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

Procedure

1. Select **AI Registry** from the left navigation pane.
2. Select the model you want to deploy.

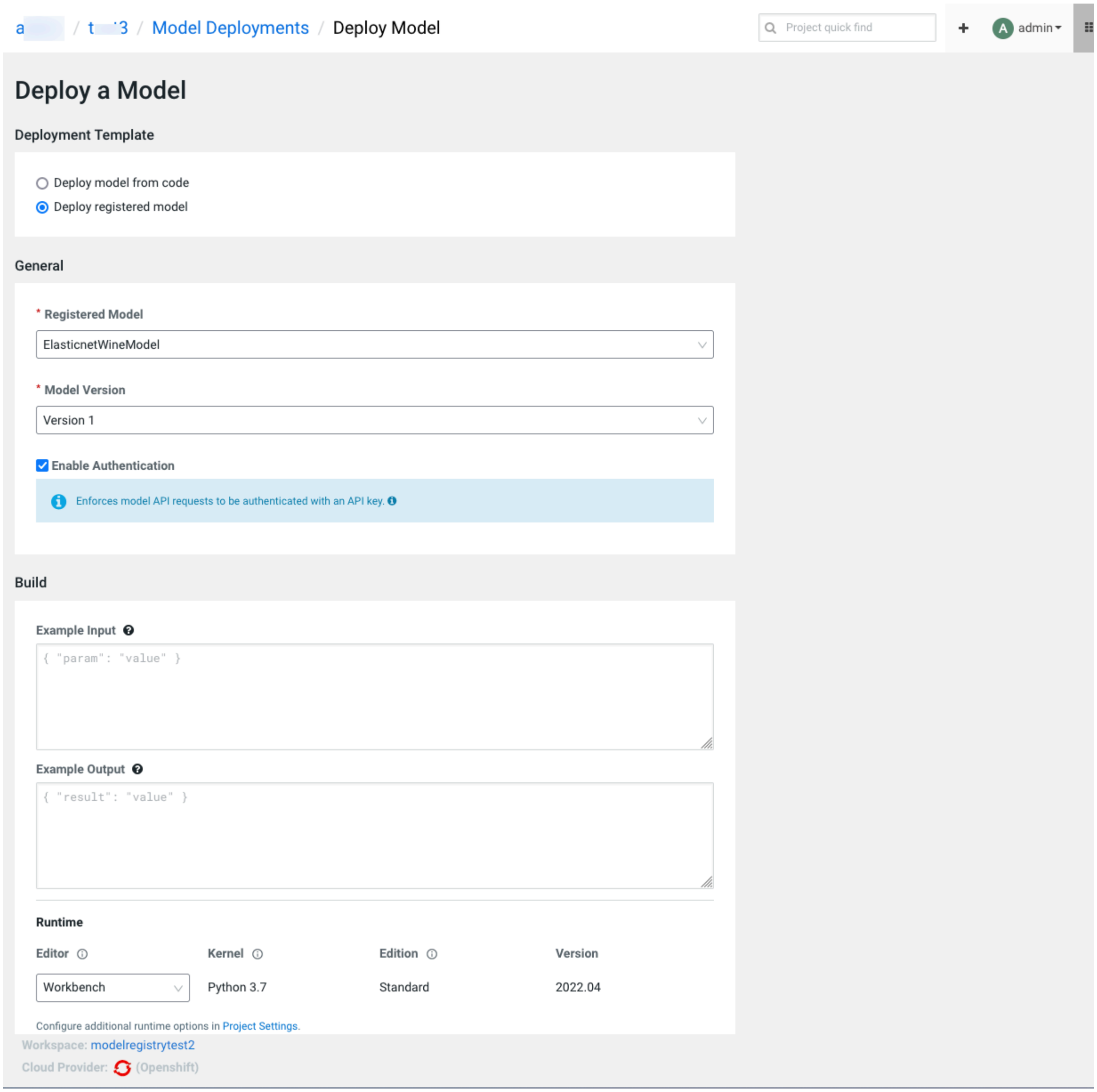
   AI Registry displays the Model Version List page.
3. Select the model version you want to deploy.

   AI 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.

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



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 Cloudera AI Registry using APIv2

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

The following example code shows how to deploy a model from the **AI 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_identifier='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 workbench to a different workbench.

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 and synchronizing the Cloudera AI Registry instance

You can view detailed information for Cloudera AI Registry.

Procedure

1. In the Cloudera Console, click the Cloudera AI tile.

   The **Cloudera AI Workbenches** page displays.
2. Select a workbench.

   The **Workbenches Home** page displays.
3. Select AI Registry from the left navigation pane.

   On the main AI 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.
4. Choose the Cloudera AI Registry you want to synchronize with the workbenches in the environment.
5. 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.
6. Click OK.

Deleting a model from Cloudera AI Registry

You can delete a model from Cloudera AI Registry through the UI or using an API call.

Deleting through the UI

1. In **AI 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 Cloudera AI 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-iylh',
   ```
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-iylh')
   ```

The model is deleted from the Cloudera AI Registry.

Disabling Cloudera AI Registry

By default, Cloudera AI Registry is enabled in Cloudera AI. You can disable Cloudera AI 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 AI Registry checkbox.

Cloudera AI Registry standalone API

You can use the standalone Cloudera AI Registry API to communicate with the Cloudera AI Registry using the REST client or CLI client.

The Cloudera AI 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 Cloudera AI Registry through the POST method

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

Cloud Platforms

Cloudera AI Registry API is available only on AWS and Azure.

API definition

The Swagger definition is available in the Cloudera AI API documentation.

Prerequisites for Cloudera AI Registry standalone API

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

Prerequisites for Cloudera AI Inference service

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

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

Clients that interact with the Cloudera AI Registry Standalone API and with model endpoints must obtain a JSON Web Token (JWT) from the Cloudera 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 Cloudera 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

Cloudera AI Registry implements role-based access control.

Users must have the following roles to create an instance of the service in a Cloudera environment:

- EnvironmentAdmin
- MLAdmin (admin user)

Registered Models can be viewed, created, deleted, and modified by users having EnvironmentUser role along with either one of the following roles:

- MLAdmin (admin user)
- MLUser

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

Related Information

Model access control

Using the REST Client

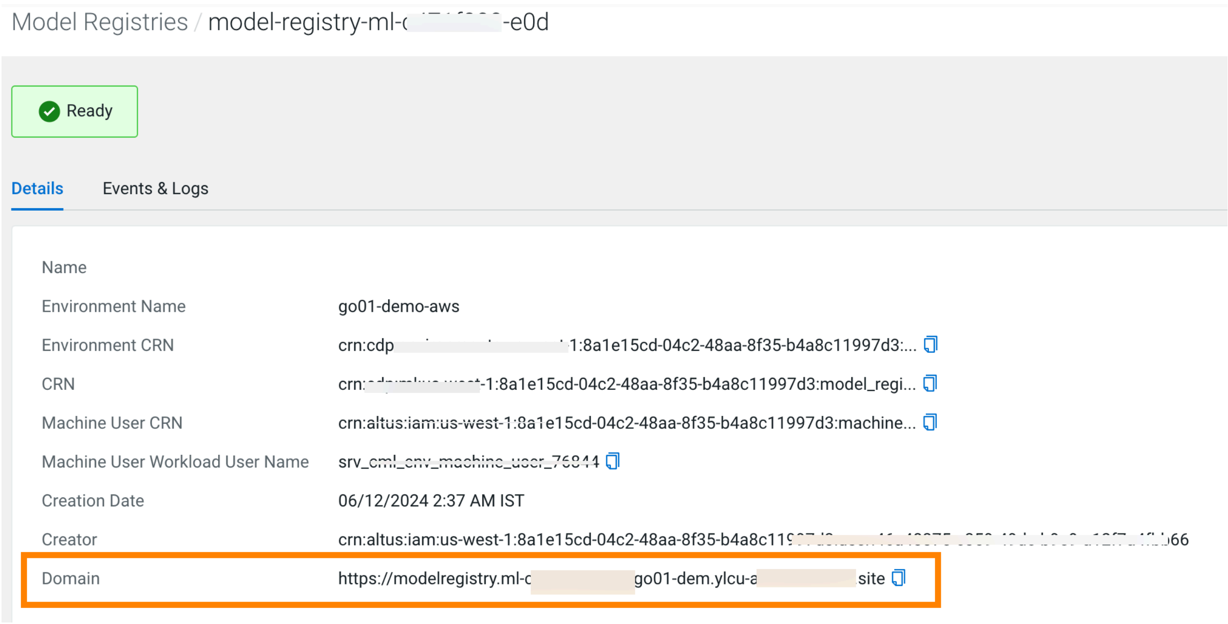
You need the domain information to use the REST client to interact with the registry.

Before you begin

To obtain the domain information, perform the following:

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

2. Click AI Registries in the left navigation menu. The AI Registries page displays.
3. Click on the name of the Cloudera AI Registry to display the Cloudera AI Registry information. The Domain name is displayed in the Details tab.



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-e1vg-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-yszl-jrt2 | jq
          {
          "created_at": "2024-04-18T15:54:24.940Z",
          "creator": {
          "user_name": "csso_cheyuanl"
          },
          "id": "fx0k-baf7-yszl-jrt2",
          "model_versions": [
          {
          "artifact_uri": "abfs://data@engmldevenvazuresan.dfs.core.windows
.net/modelregistry/fx0k-baf7-yszl-jrt2/y8d8-qluc-00md-h2pw/model.tar.gz",
          "created_at": "2024-04-18T15:54:24.942Z",
          "model_id": "fx0k-baf7-yszl-jrt2",
          "status": "READY",
          "tags": null,
```

```
            "updated_at": "2024-04-18T15:54:24.942Z",
            "user": {
            "user_name": "csso_cheyuanl"
            },
            "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-yszl-jrt2/versions/1 | jq
            {
            "artifact_uri": "abfs://data@engmldevenvazuresan.dfs.core.windows.
net/modelregistry/fx0k-baf7-yszl-jrt2/y8d8-qluc-00md-h2pw/model.tar.gz",
            "created_at": "2024-04-18T15:54:24.942Z",
            "model_id": "fx0k-baf7-yszl-jrt2",
            "status": "READY",
            "tags": null,
            "updated_at": "2024-04-18T15:54:24.942Z",
            "user": {
            "user_name": "csso_cheyuanl"
            },
            "version": 1
            }
```

Patch a Model

```
curl -XPATCH -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/mod
els/fx0k-baf7-yszl-jrt2 -d ‘{
  “visibility”: “public”
}’
```

Patch a Model Version

```
curl -XPATCH -s -H "Authorization: Bearer ${CDP_TOKEN}" ${DOMAIN}/api/v2/mod
els/fx0k-baf7-yszl-jrt2/version/1 -d ‘{
  “tags”: [{“key”: “k1”, “value”: “v1”}, “key”: “k2”, “value”: “v2”}]
}’
```

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
```

Cloudera AI Registry CLI Client

Cloudera AI Registry client is a command line tool (CLI) that can be used to interact to a Cloudera AI Registry server. It can be downloaded from any Cloudera AI Registry server <domain>/apiv2/cli/<os>. Here, <os> is either Linux, Darwin for Mac, or Windows based on the operating system the Cloudera AI 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 string          config file path
      --debug                  output debug logs
      --dry-run                do not send the request to server
  -h, --help                   help for modelregistrycli
      --hostname string        hostname of the service (default "localhost")
      --scheme string          Choose from: [http] (default "http")
```

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": "prajjwal1/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 op
erations 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:9
000/8fts-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 reque
st 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,"u
pdated_at":"2024-04-03T18:08:43.130Z","visibility":"public"}
```

Known issues with Cloudera AI Registry standalone API

These are some of the known issues you might run into while using Cloudera AI Registry standlone 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:  ohlfw0olaadg/ea-participants/llama-2-7b-chat:LLAMA-2-
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: 5)
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 the command 'ngc ver
sion upgrade'


2024/04/23 16:53:45 Failed to download NGC model repo to local folder:  exit status 1
```

Retry the model import request again.

Model import failure

You can download the models concurrently only if their combined size is below approximately 400 GB. Exceeding this limit may result in import failures and unexpected behavior.

Request Throttling

Currently, there is no request throttling mechanism implemented. As a result, excessive concurrent requests may lead to model import failures. To minimize the risk, it is recommended to limit concurrent requests to a maximum of 5, which is considered a safe threshold.

Model Import progress indicator

A progress bar is not available for model imports. For reference, importing a 70 GB model typically takes approximately 1 hour. Users should plan accordingly and monitor the process through alternative options, if necessary.

Related Information

Debugging the model import failure

Manually Updating Cloudera AI Registry Configuration

Troubleshooting issues with Cloudera AI Registry API

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

Cloudera AI Inference service cannot discover Cloudera AI Registry

Learn about scenarios or issues that may be resolved by using the steps in the *Manually updating Cloudera AI Registry configuration* solution.

- ML Serving installed before Cloudera AI Registry

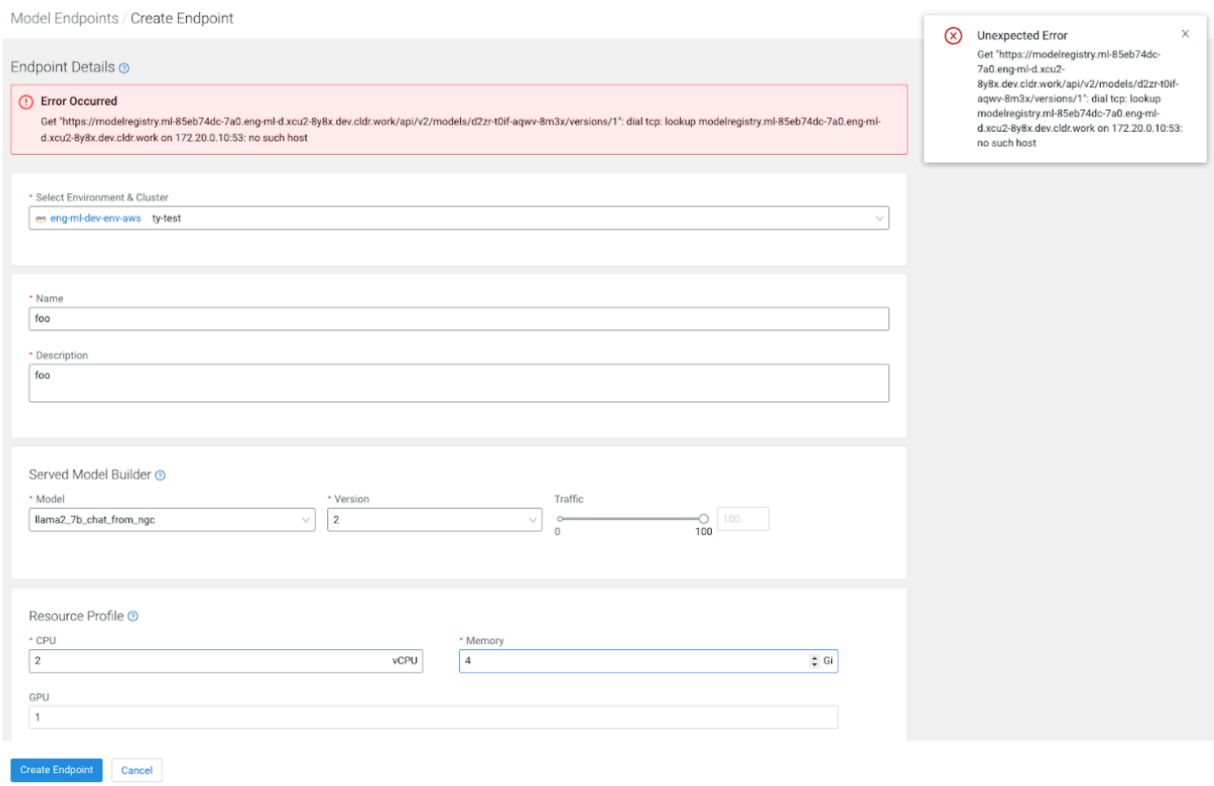
  If the Cloudera AI Registry has not been created yet, when ML Serving is installed it will not be able to generate the ConfigMap.

- Cloudera AI Registry upgraded after ML Serving installed

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

- Cloudera AI Registry certificate expired

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



Manually updating Cloudera AI Registry configuration

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

Important: You are required to access multiple clusters when performing the below steps. For information configuring and switching KubeConfig between multiple clusters, see Configure Access to Multiple Clusters

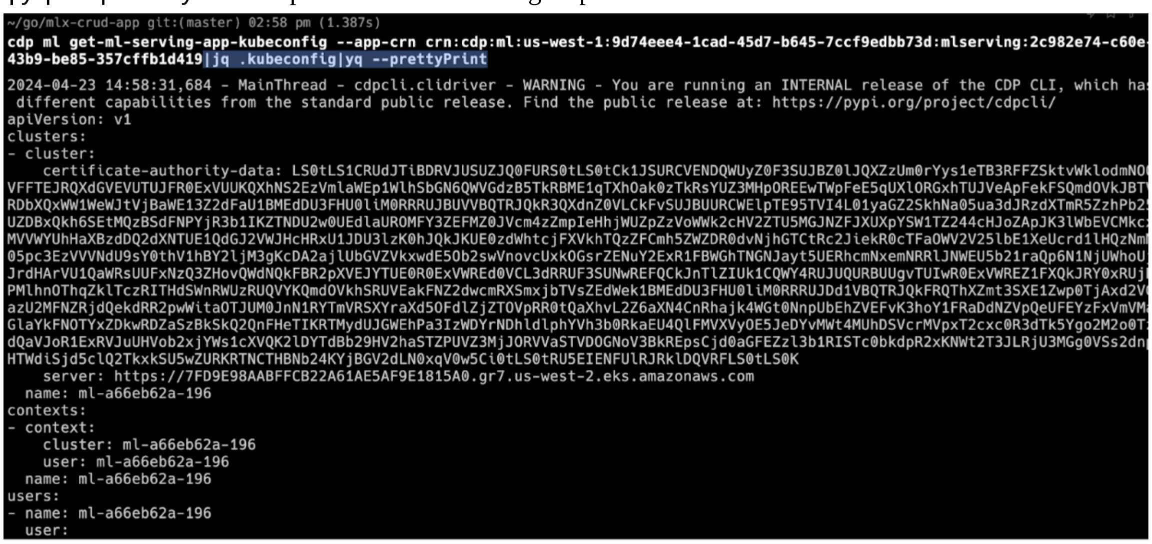
1. Get KubeConfig for ML Serving Cluster.
   a. In a terminal where Cloudera CLI is installed, run the cdp ml list-ml-serving-apps command.
   b. Find the appCrn of the ML Serving instance you wish to update.

   c. Retrieve the KubeConfig with this command:

```
cdp ml get-ml-serving-app-kubeconfig --app-crn <YOUR-APP-CRN> |jq .kubec
onfig|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 kKubeConfig response into a format both machine and user-readable.



   d. Direct the output to the standard KubeConfig location:

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

   Make sure this is the active KubeConfig by running the `kubectl config view` command. The output of the above command shall now show the KubeConfig data returned above.

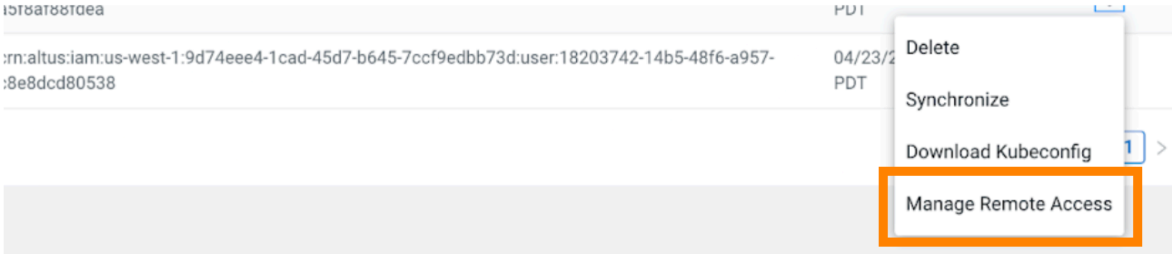
2. Grant your user ML Serving App access.

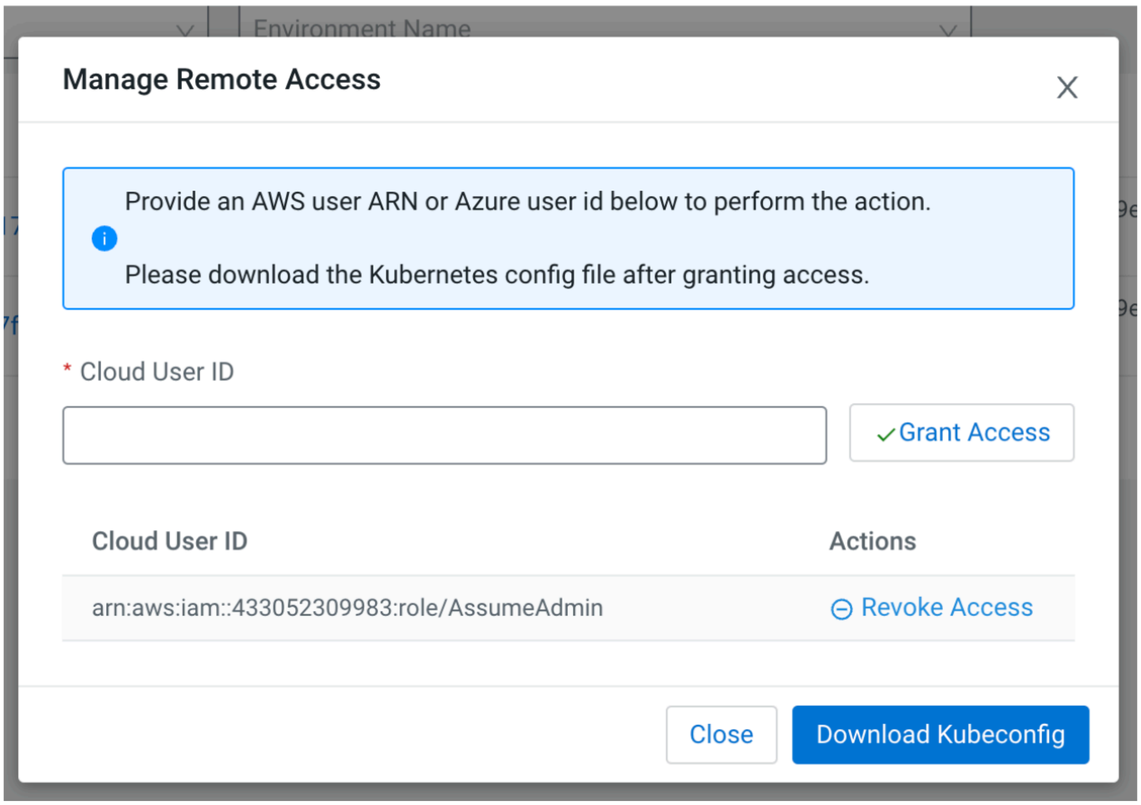
   Add your cloud provider user identifier to the ML Serving App access control list. Use the following command, with your ARN, or other cloud provider user identifier.

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





   b. After your user ARN has been granted access to the Cloudera AI Registry, get the ConfigMap data in the following way:

   - Download the Cloudera AI Registry KubeConfig.
   - Set the Cloudera AI Registry KubeConfig (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 Cloudera AI Registry.
   - `cdp ml list-model-registries`: The response will contain the domain for the updated Cloudera AI Registry.

   c. Copy the entire rootca.pem output from the command.

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

```
Data
====
rootca.pem:
----
-----BEGIN CERTIFICATE-----
```


```
MIIFmDCCA4CgAwIBAgIQU9C87nMpOIFKYpfvOHFHFDANBgkqhkiG9w0BAQsFADBm
MQswCQYDVQQGEwJVUzEzMDEGA1UEChMqKFNUQUdJTkcpIEludGVybmV0IFNlY3Vy
aXR5IFJlc2VhcmNoIEdyb3VwMSIwIAYDVQQDExkoU1RBR0lORykgUHJldGVuZCBQ
ZWFyIFgxMB4XDTE1MDYwNDExMDQzOFoXDTM1MDYwNDExMDQzOFowZjELMAkGA1UE
BhMCVVMxMzAxBgNVBAoTKihTVEFHSU5HKSBJbnRlcm5ldCBTZWN1cml0eSBSZXNl
YXJjaCBHcm91cDEiMCAGA1UEAxMZKFNUQUdJTkcpIFByZXRlbmQgUGVhciBYMTCC
AiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBALbagEdDTa1QgGBWSYkyMhsc
ZXENOBaVRTMX1hceJENgsL0Ma49D3MilI4KS38mtkmdF6cPWnL++fgehT0FbRHZg
jOEr8UAN4jH6omjrbTD++VZneTsMVaGamQmDdFl5g1gYaigkkmx8OiCO68a4QXg4
wSyn6iDipKP8utsE+x1E28SA75HOYqpdrk4HGxuULvlr03wZGTIf/oRt2/c+dYmD
oaJhge+GOrLAEQByO7+8+vzOwpNAPEx6LW+crEEZ7eBXih6VP19sTGy3yfqK5tPt
TdXXCOQMKAp+gCj/VByhmIr+0iNDC540gtvV303WpcbwnkkLYC0Ft2cYUyHtkstO
fRcRO+K2cZozoSwVPyB8/J9RpcRK3jgnX9lujfwA/pAbP0J2UPQFxmWFRQnFjaq6
rkqbNEBgLy+kFL1NEsRbvFbKrRi5bYy2lNms2NJPZvdNQbT/2dBZKmJqxHkxCuOQ
FjhJQNeO+Njm1Z1iATS/3rts2yZlqXKsxQUzN6vNbD8KnXRMEeOXUYvbV4lqfCf8
mS14WEbSiMy87GB5S9ucSV1XUrlTG5UGcMSZOBcEUpisRPEmQWUOTWIoDQ5FOia/
GI+Ki523r2ruEmbmG37EBSBXdxIdndqrjy+QVAmCebyDx9eVEGOIpn26bW5LKeru
mJxa/CFBaKi4bRvmdJRLAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBS182Xy/rAKkh/7PH3zRKCsYyXDFDANBgkqhkiG
9w0BAQsFAAOCAgEAncDZNytDbrrVe68UT6py1lfF2h6Tm2p8ro42i87WWyP2LK8Y
nLHC0hvNfWeWmjZQYBQfGC5c7aQRezak+tHLdmrNKHkn5kn+9E9LCjCaEsyIIn2j
qdHlAkepu/C3KnNtVx5tW07e5bvIjJScwkCDbP3akWQixPpRFAsnP+ULx7k0aO1x
qAeaAhQ2rgo1F58hcflgqKTXnpPM02intVfiVVkX5GXpJjK5EoQtLceyGOrkxlM/
sTPq4UrnypmsqSagWV3HcUlYtDinc+nukFk6eR4XkzXBbwKajl0YjztfrCIHOn5Q
CJL6TERVDbM/aAPlv8kJ1sWGLuvvWYzMYgLzDul//rUF10gEMWaXVZV51KpS9DY/
```


The domain of the new Cloudera AI 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://pypi.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-ebe71d0dae2d",
            "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 Cloudera Manager 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

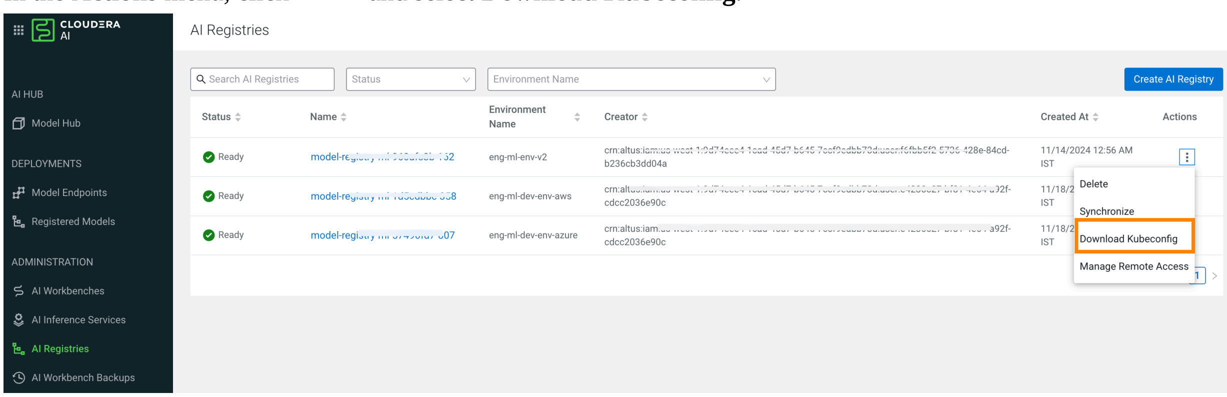
To debug errors that occurred on the Cloudera AI Registry server, you can access the logs found in the API v2 pod.

About this task

Access logs from Cloudera AI Registry Kubernetes cluster.

You can obtain the kubeconfig for the Cloudera AI Registry cluster.

1. In the **Cloudera** console, click the **Cloudera AI** tile.
2. Click AI Registries in the left navigation menu. The AI Registries page displays.
3. In the **Actions** menu, click  and select **Download Kubeconfig**.



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 Cloudera AI Workbench*. After the kubeconfig is set up, run the following `kubectl` command to get logs for the Cloudera AI Registry pod:

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

Related Information

Granting Remote Access to Cloudera AI Workbenches

Importing a Hugging Face Model (Technical Preview)

If your desired Hugging Face model is unavailable on the Model Hub page, you can import those models from the *Hugging Face* website. After you import the model, the newly imported model will be listed on the Registered Models page.

Note: This feature is in Technical Preview and not recommended for production deployments. Cloudera recommends that you try this feature in test or development environments.

Procedure

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

   The **Cloudera AI Workbenches** page displays.
2. Click **Registered Models** under **Deployments** in the left navigation menu.

   The **Registered Models** page displays. The page lists all the models of different Cloudera AI Registries along with the associated metadata.

3. Click Import Model. The Import Model page displays.


Import Model
Technical Preview - Import Hugging Face Models
This feature is in Technical Preview, so some models may not fully integrate with Cloudera AI Inference Service.
* AI Registry
Select AI Registry
* Name
Visibility
Public
Private
* Repository ID
Hugging Face Token
Enter your Hugging Face Token
Description
Version Notes
Cancel
Import

4. In the AI Registry drop-down list, select the AI registry to which you want to import the model.
5. In the Name field, enter a new name for the model you are importing.
6. Select the Visibility as Public or Private. If you select Public, the model is available for other users. If you select Private, the model is displayed on the Registered Models page only for the user who imported it.
7. In the Repository ID field, enter the ID of the Hugging Face model. You can obtain the ID of a model from the Hugging Face website.
8. In the Hugging Face Token field, enter the token obtained from the Hugging Face website.
9. In the Description field, enter a description for the model.
10. In the Version Notes field, enter notes about this version of the model.
11. Click Import.

Results

You can view this newly imported model on the Registered Models page.

Creating and deploying a Model

Using Cloudera AI, you can create any function within a script and deploy it to a REST API. In a Cloudera AI project, this is typically a predict function that accepts an input and returns a prediction based on the model's parameters.

Using Cloudera AI, 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 Cloudera AI 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

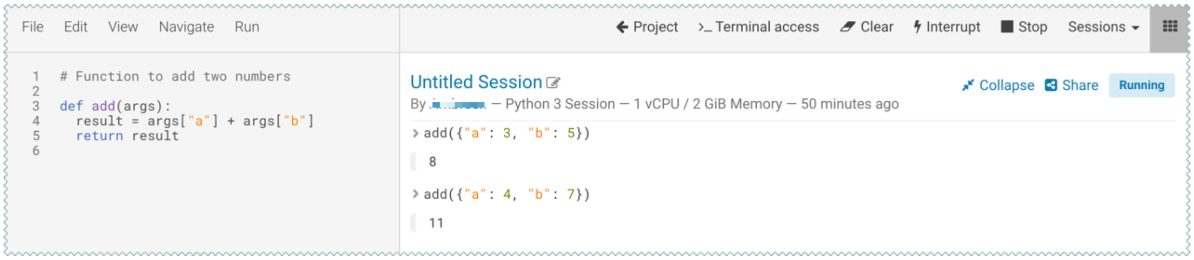
   ```
   import cml.models_v1 as models

   @models.cml_model
   def add2(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.

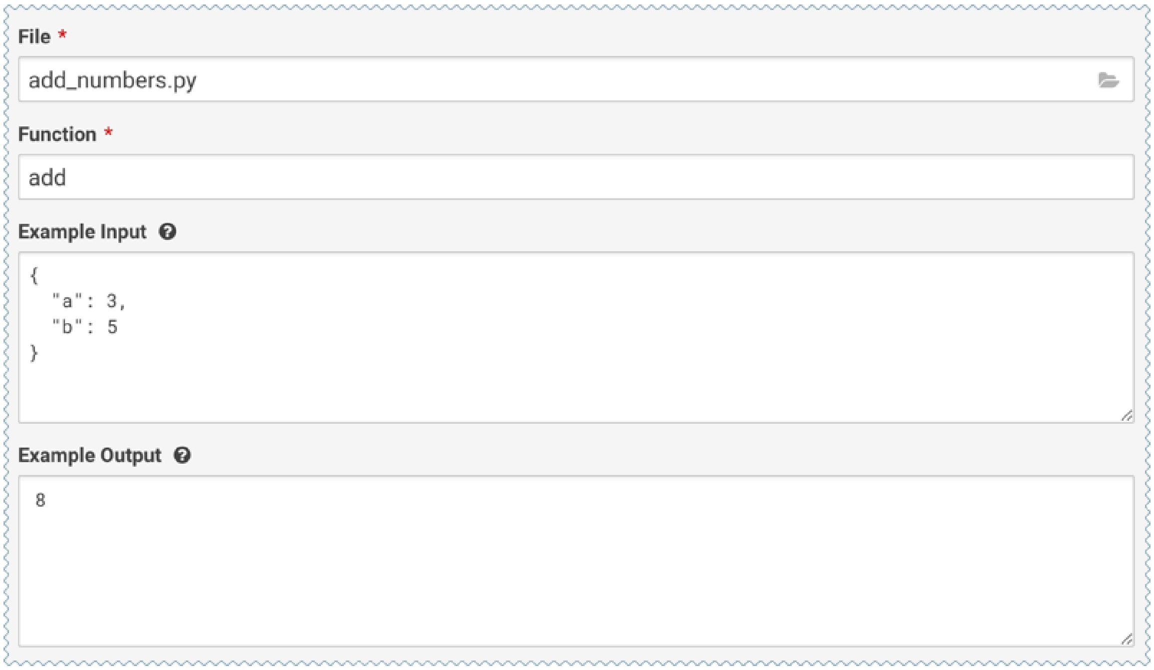
b. Click Models New Model .

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

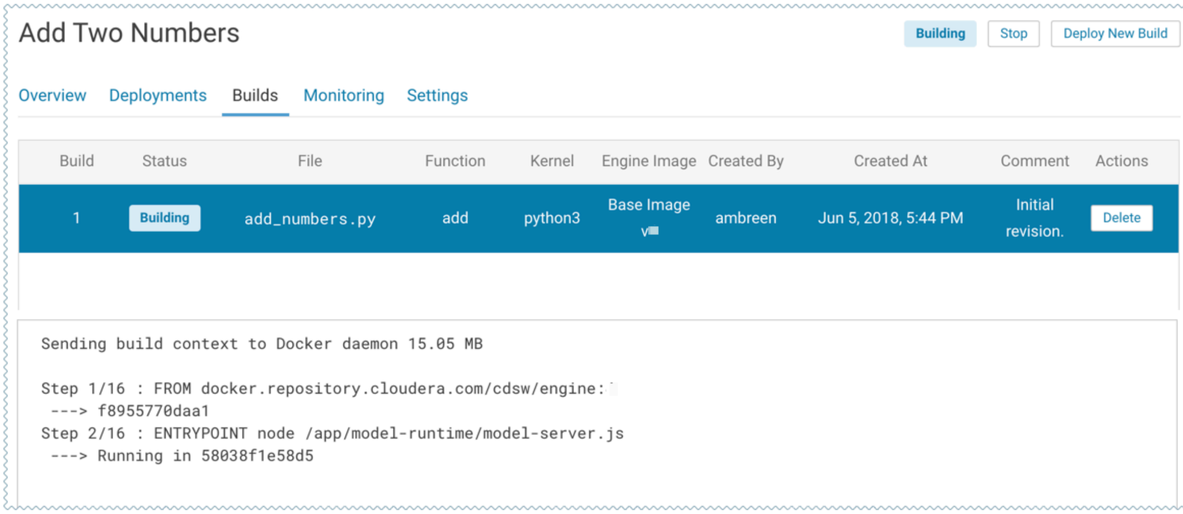


f. Select the resources needed to run this model, including any replicas for load balancing. To specify the maximum number of replicas in a model deployment, go to Site Administration Settings Models . The default is 9 replicas, and up to 199 can be set.

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.

6. Click on the model to go to its Overview page. Click Builds to track realtime progress as the model is built and deployed. This process essentially creates a Docker container where the model will live and serve requests.



7. Once the model has been deployed, go back to the model Overview page and use the Test Model widget to make sure the model works as expected.

   If you entered example input when creating the model, the Input field will be pre-populated with those values. Click Test. The result returned includes the output response from the model, as well as the ID of the replica that served the request.

   Model response times depend largely on your model code. That is, how long it takes the model function to perform the computation needed to return a prediction. It is worth noting that model replicas can only process one request at a time. Concurrent requests will be queued until the model can process them.

For Cloudera AI APIv2

Create and deploy a model using the Models API as instructed in this example:

This example demonstrates the use of the Models API. To run this example, first do the following:

1. Create a project with the Python template and a legacy engine.
2. Start a session.
3. Run !pip3 install sklearn
4. Run fit.py

The example script first obtains the project ID, then creates and deploys a model.

```
projects = client.list_projects(search_filter=json.dumps({"name": "<your
 project name>"}))
project = projects.projects[0] # assuming only one project is returned by
 the above query
model_body = cmlapi.CreateModelRequest(project_id=project.id, name="Demo
 Model", description="A simple model")
model = client.create_model(model_body, project.id)
model_build_body = cmlapi.CreateModelBuildRequest(project_id=project.id,
 model_id=model.id, file_path="predict.py", function_name="predict", ker
nel="python3")
model_build = client.create_model_build(model_build_body, project.id, mod
el.id)
while model_build.status not in ["built", "build failed"]:
 print("waiting for model to build...")
time.sleep(10)
 model_build = client.get_model_build(project.id, model.id, model_build
.id)
if model_build.status == "build failed":
```

```
 print("model build failed, see UI for more information")
 sys.exit(1)
print("model built successfully!")
model_deployment_body = cmlapi.CreateModelDeploymentRequest(project_id=p
roject.id, model_id=model.id, build_id=model_build.id)
model_deployment = client.create_model_deployment(model_deployment_body,
project.id, model.id, build.id)
while model_deployment.status not in ["stopped", "failed", "deployed"]:
 print("waiting for model to deploy...")
 time.sleep(10)
 model_deployment = client.get_model_deployment(project.id, model.id, m
odel_build.id, model_deployment.id)
if model_deployment.status != "deployed":
 print("model deployment failed, see UI for more information")
 sys.exit(1)
print("model deployed successfully!")
```

Related Information

Example models with PBJ Runtimes

Usage guidelines for deploying models with Cloudera AI

Consider these guidelines when deploying models with Cloudera AI.

Model Code

Models in Cloudera AI 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.

Notice:

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 guidelines on *Monitoring Active Models* 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 AI 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 AI 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 AI 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 shall 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 AI 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 with Model Builds and Deployed Models

- Known Issues with Model Builds and Deployed Models
  - Re-deploying or re-building models results in model downtime (usually brief).
  - Re-starting Cloudera AI 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 checkpoi
    nt of current source: [Unable to push sources to git server: ...
    ```

    To work around this, rename the .git directory (for example, NO.git) and re-build the model.

    Cloudera Bug: DSE-4657
  - JSON requests made to active models should not be more than 5 MB in size. This is because JSON is not suitable for very large requests and has high overhead for binary objects such as images or video. Call the model with a reference to the image or video, such as a URL, instead of the object itself.
  - Any external connections, for example, a database connection or a Spark context, must be managed by the model's code. Models that require such connections are responsible for their own setup, teardown, and refresh.
  - Model logs and statistics are only preserved so long as the individual replica is active. Cloudera AI 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 Cloudera AI 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 with Model Builds and Deployed Models
  - Scala models are not supported.
  - Spawning worker threads are not supported with models.
  - Models deployed using Cloudera AI 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 AI 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 AI 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 AI will keep trying to contact a free replica for 30 seconds. If no replica is available, Cloudera AI 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 Administrator user, open a CLI.
2. At the prompt, execute the following command. Substitute <value> with the number of seconds to set.

   ```
   kubectl set env deployment model-proxy MODEL_REQUEST_TIMEOUT_SECONDS=<va
   lue> -n mlx
   ```

   This edits the `kubeconfig` file and sets a new value for the timeout duration.

Related Information

Creating and deploying a Model

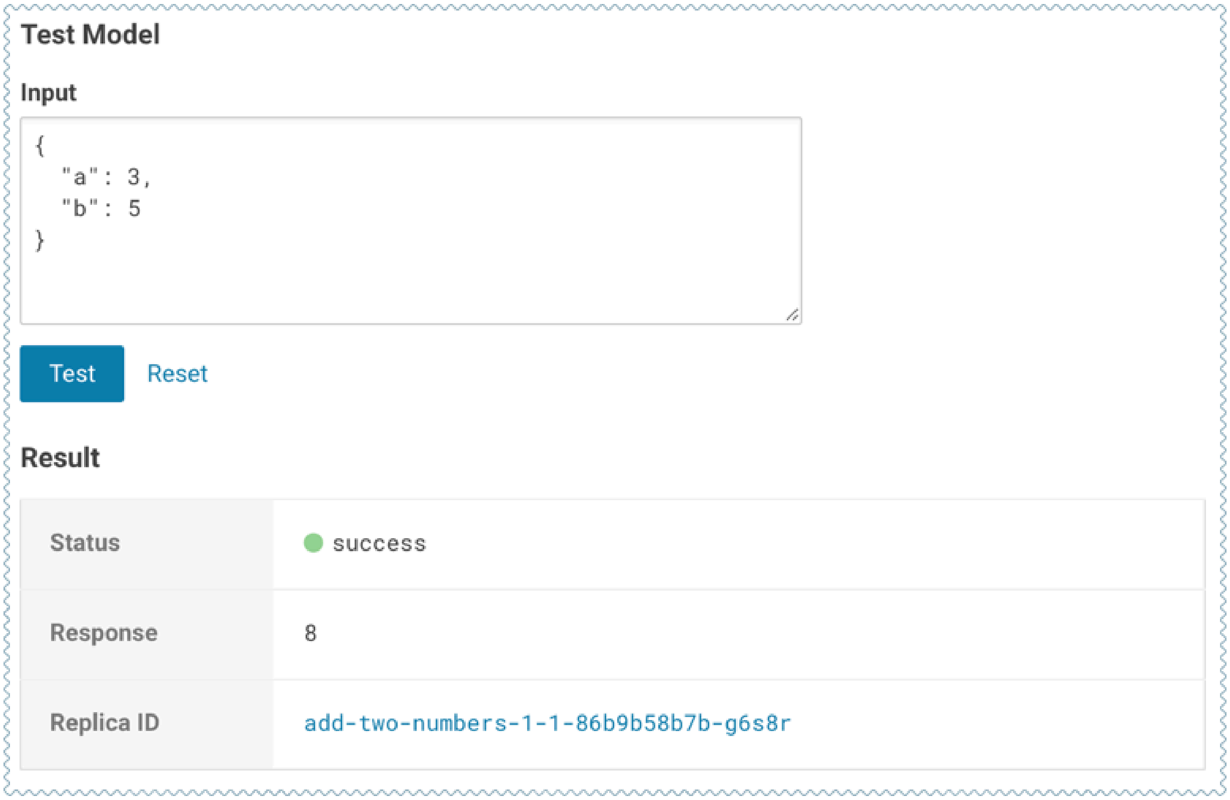
Workflows for Active Models

Testing calls to a Model

Cloudera AI provides two ways to test calls to a model:

- Test Model Widget

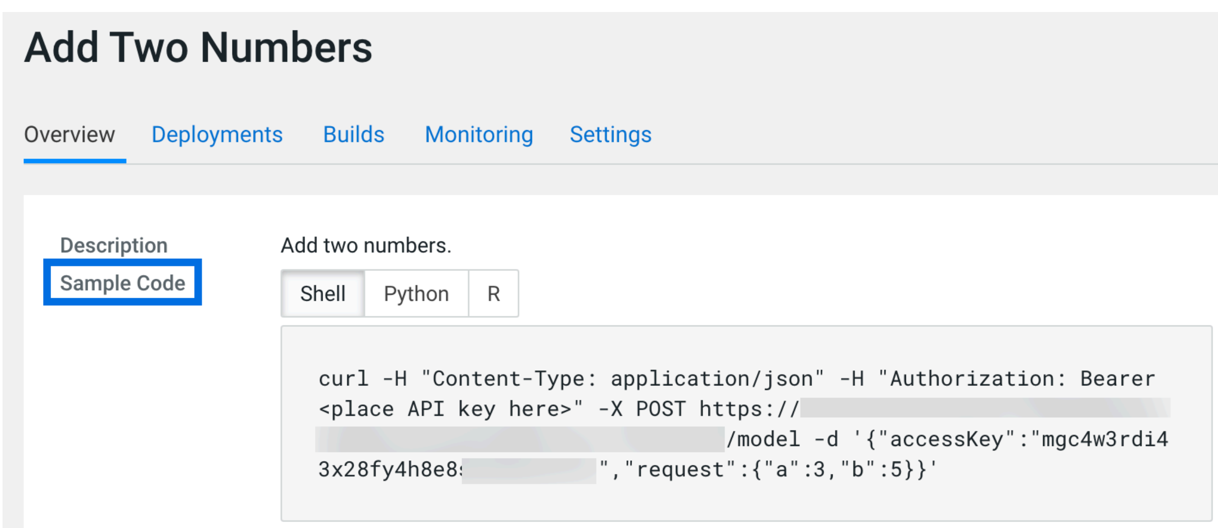
  On each model's Overview page, Cloudera AI provides a widget that makes a sample call to the deployed model to ensure it is receiving input and returning results as expected.



- Sample Request Strings

  On the model Overview page, Cloudera AI 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.



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

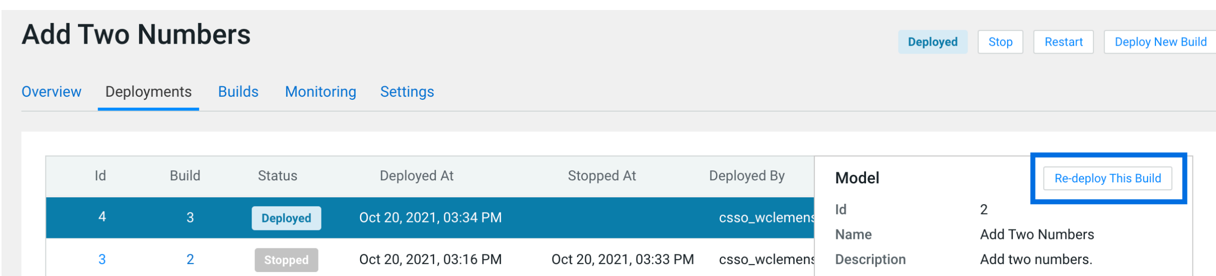
Re-deploying a model involves re-publishing a previously-deployed model in a new serving environment - this is, with an updated number of replicas or memory/CPU/GPU allocation. For example, circumstances that require a re-deployment might include:

- 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 AI 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.



4. Modify the model serving environment as needed.
5. Click Deploy Model.

Deploy a new build for a Model

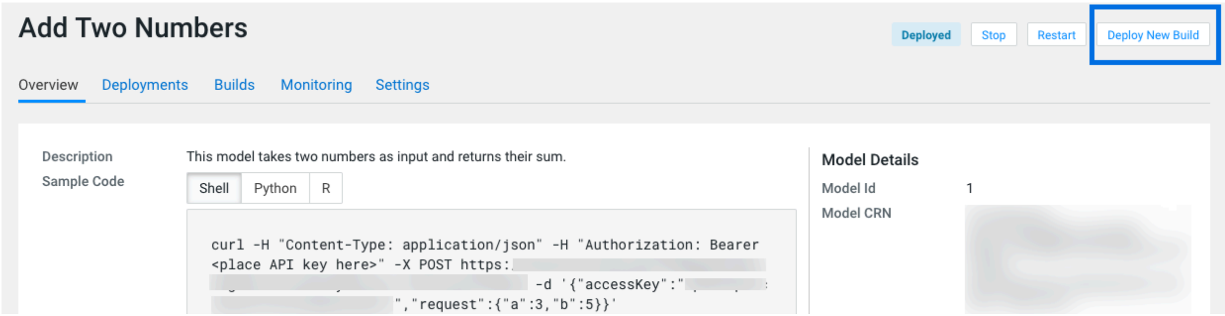
Deploying a new build for a model involves both, re-building the Docker image for the model, and deploying this new build. Note that this is not required if you only need to update the resources allocated to the model. As an example, changes that require a new build might include:

- Code changes to the model implementation.
- Renaming the function that is used to invoke the model.

Warning: Currently, Cloudera AI 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 AI hosts. Depending on network speeds, your model may spend some time in this stage.

Deploying

If you see issues occurring when Cloudera AI 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 AI 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 helps keeping 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 the deployments are not stopped, the active models will continue serving requests and consuming resources even though they do not show up in Cloudera AI 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 AI.

   You can also delete specific builds from a model's history by going to the model's Overview Build page.

Configuring model metrics payload limit

Model metrics have a configuration that restricts model request payload to 100 KB. You can increase the payload size if required.

Procedure

1. Convert the payload size to bytes.

   For example, 100 KB (Kilobytes) = 100 * 1024 bytes = 102400 bytes.
2. Encode the values into the Base64 format.

   For example, 20 MB in Bytes is 20000000, to convert it to Base64 value, run the following command on the terminal:

   ```
   echo -n "20000000" | base64
   ```

   The output would be MjAwMDAwMDA=. Here, MjAwMDAwMDA= is the Base64 encoded value.
3. Edit the existing Secret object to specify the model request payload size.

   ```
   kubectl edit secret model-metrics-config -n mlx
   ```

   This opens your default editor and allows you to update the Base64 encoded values.
4. Locate the max.request.size.bytes field and update it with the Base64 encoded value.

   ```
   max.request.size.bytes: [***Base64-encoded-value***]
   ```

   For example, substitute the above 20 MB Base64 encoded value:

   ```
   max.request.size.bytes: MjAwMDAwMDA=
   ```
5. Save and close the editor.
6. Restart the model metrics pod by deleting the pod.

   ```
    kubectl delete pod modelmetrics [***pod nam-ext***] -n mlx
   ```

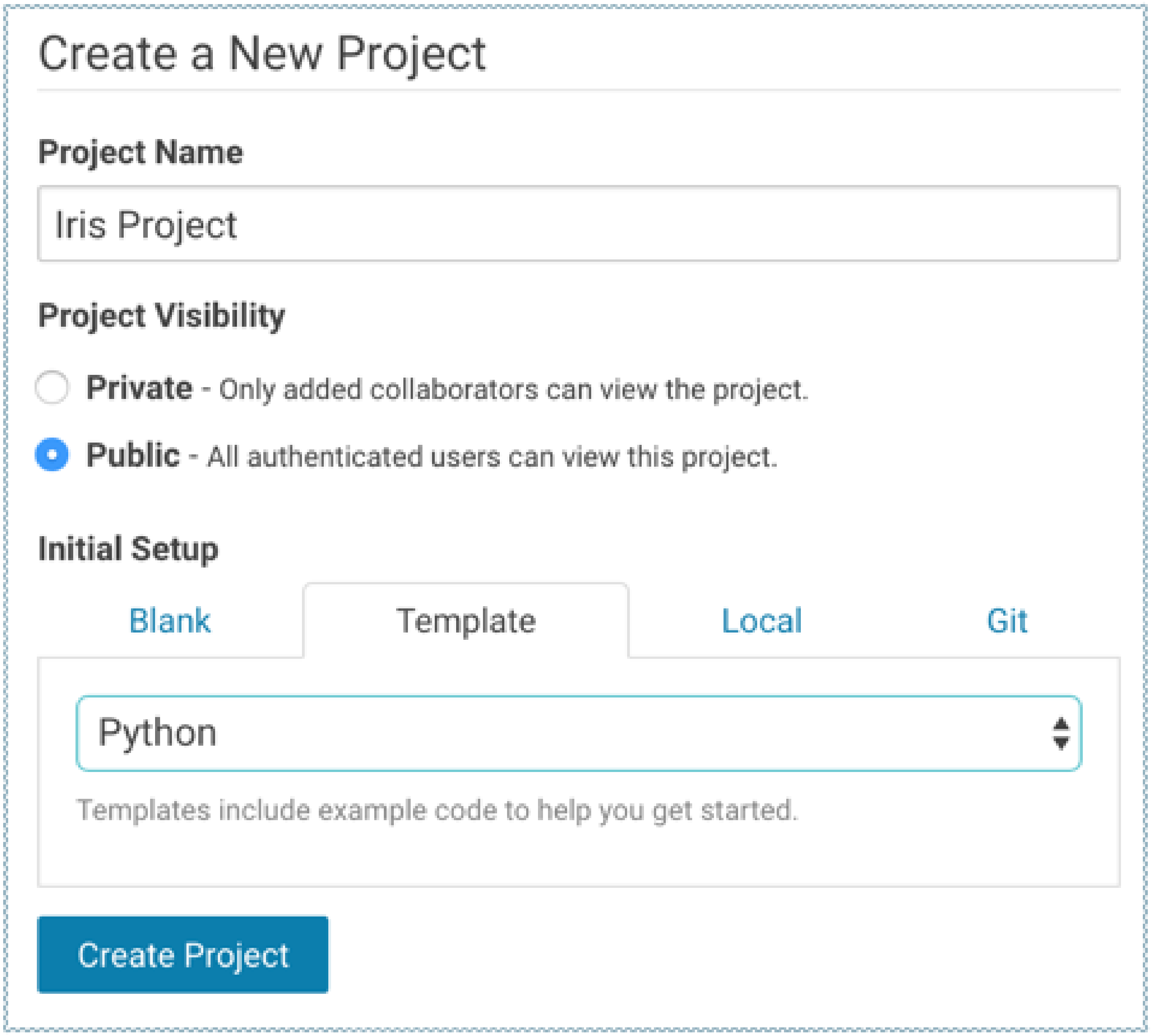
   This will force Kubernetes to restart the pod with the updated configuration.

Example - Model training and deployment (Iris)

This topic uses Cloudera AI'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 AI.

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 AI. First, create a new project from the Python template:



Once you have 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.

Training the Model

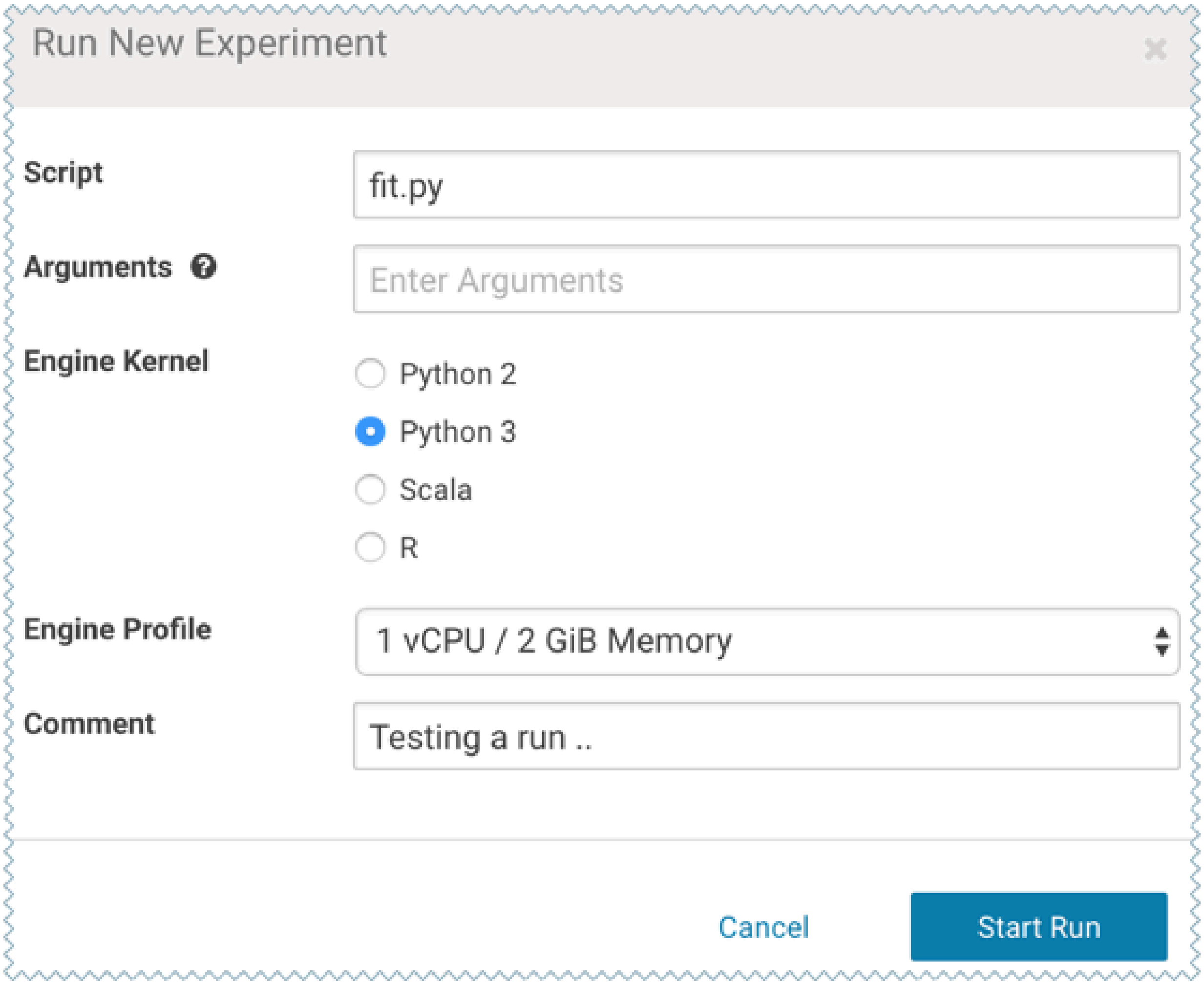
This topic shows you how to run experiments and develop a model using the fit.py file.

About this task

The fit.py script tracks metrics, mean squared error (MSE) and R2, to help compare the results of different experiments. It also writes the fitted model to a model.pkl file.

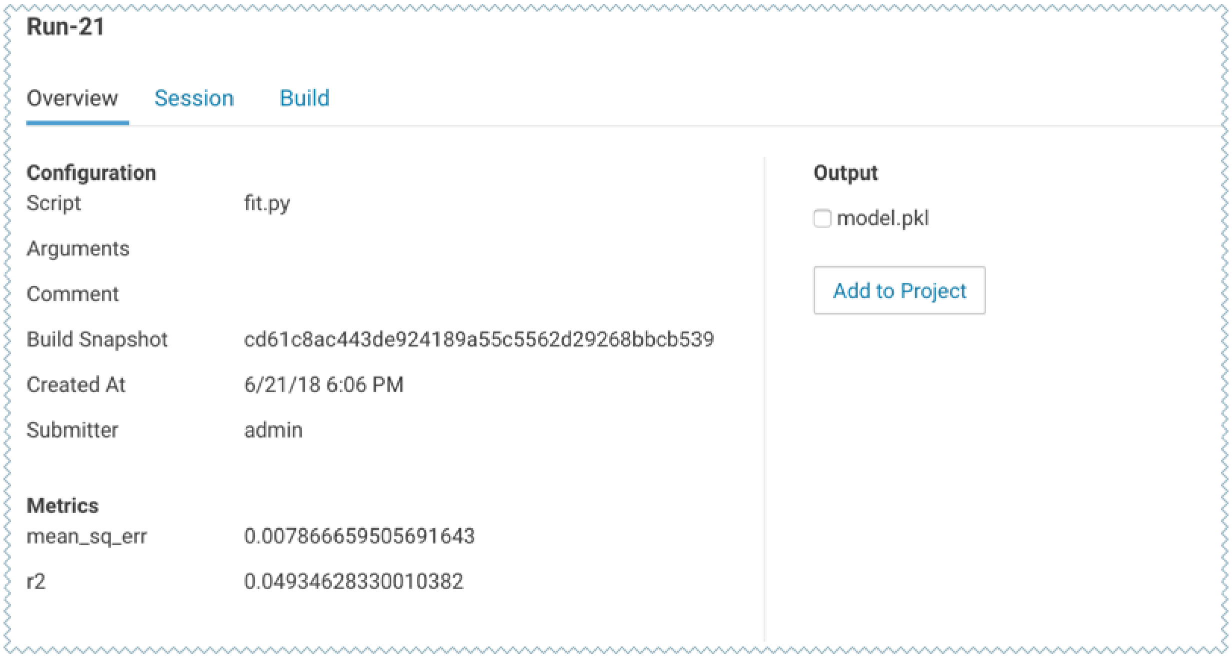
Procedure

1. Navigate to the Iris project's Overview Experiments page.
2. Click Run Experiment.
3. Fill out the form as follows and click Start Run. Make sure you use the Python 3 kernel.



4. The new experiment shall now display on the Experiments table. Click on the Run ID to go to the experiment's Overview page. The Build and Session tabs display realtime progress as the experiment builds and executes.

5. Once the experiment has completed successfully, go back to its Overview page. The tracked metrics show us that our test set had an MSE of ~0.0078 and an R2 of ~0.0493. For the purpose of this demo, let's consider this an accurate enough model to deploy and use for predictions.



6. Once you have finished training and comparing metrics from different experiments, go to the experiment that generated the best model. From the experiment's Overview page, select the model.pkl file and click Add to Project.
This saves the model to the project filesystem, available on the project's Files page. We will now deploy this model as a REST API that can serve predictions.

Deploying the Model

This topic shows you how to deploy the model using the predict.py script from the Python template project.

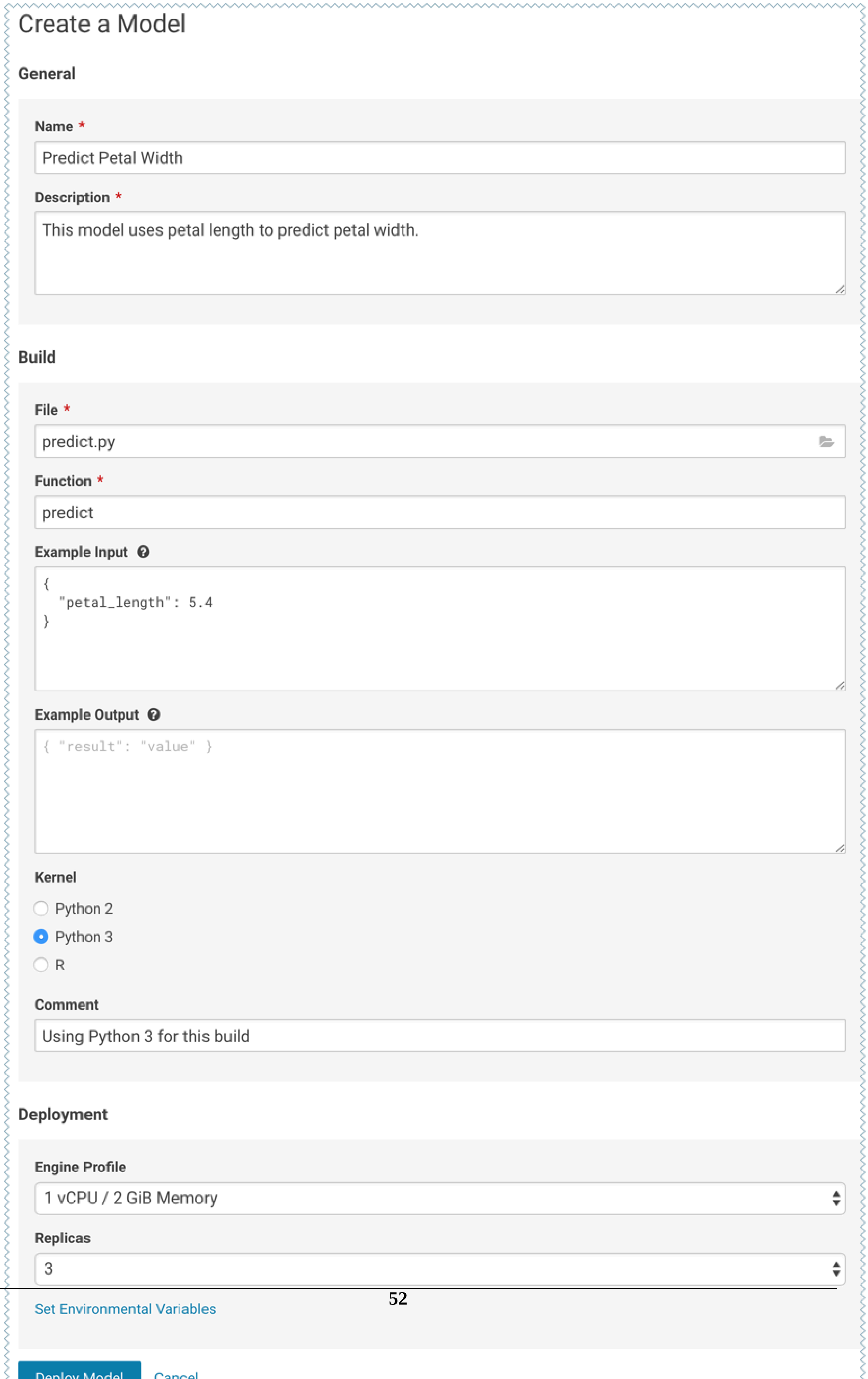
About this task

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.

Securing Models

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

Important: Cloudera on 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 on cloud environments. Since Cloudera on cloud runs in a customer's cloud account, Security and Compliance is a shared responsibility between Cloudera and its on cloud customers. For more information, see *Cloudera's Shared Responsibility Model*.

Related Information

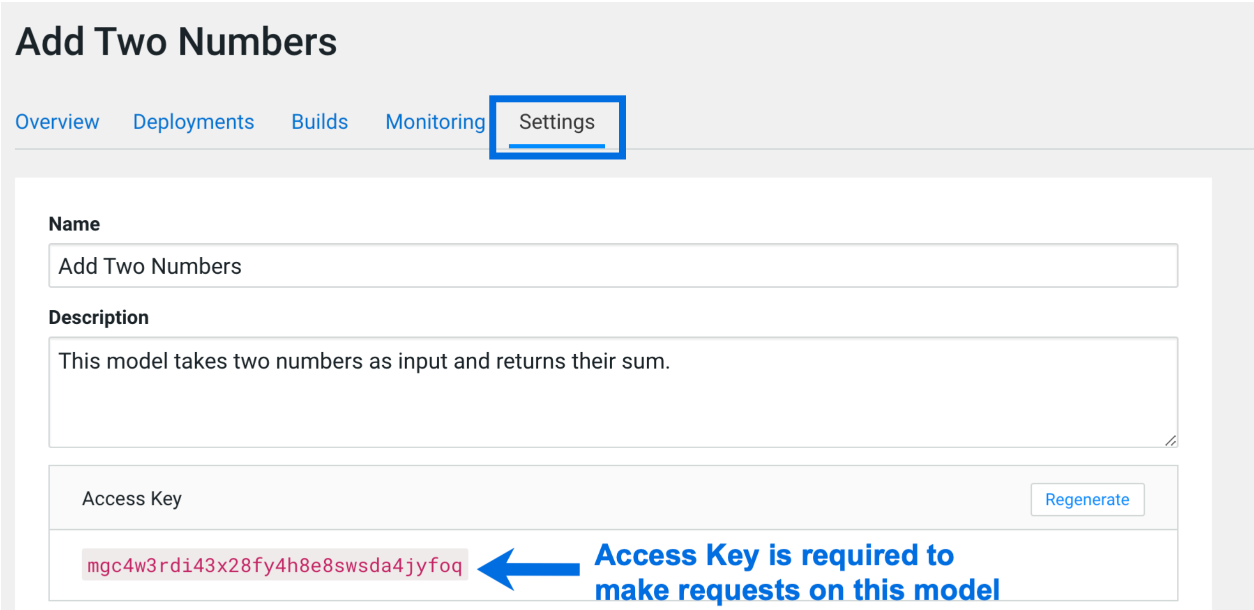
Cloudera's Shared Responsibility Model

Access Keys for Models

Each model in Cloudera AI has a unique access key associated with it. This access key is a unique identifier for the model.

Models deployed using Cloudera AI are not public. In order to call an active model your request must include the model's access key for authentication (as demonstrated in the sample calls above).

To locate the access key for a model, go to the model Overview page and click Settings.



Important:

Only one access key per model is active at any time. If you regenerate the access key, you will need to re-distribute this access key to users/applications using the model.

Alternatively, you can use this mechanism to revoke access to a model by regenerating the access key. Anyone with an older version of the key will not be able to make calls to the model.

API Key for Models

You can prevent unauthorized access to your models by specifying an API key in the **Authorization** header of your model HTTP request. This topic covers how to create, test, and use an API key in Cloudera AI.

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 AI:

- 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 cannot 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 AI.
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 administrator user can access the list of all the users who are accessing the workbench and can delete the API keys for a user.

About this task

To manage users and their keys:

Procedure

1. Sign in to Cloudera AI as an administrator 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 workbench 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 .

Model Governance

To capture and view centralized information about your ML projects, models, and builds in Apache Atlas (Data Catalog) for a specific environment, governance must be enabled.

Enabling model governance

You must enable governance to capture and view information about your ML projects, models, and builds centrally from Apache Atlas (Data Catalog) for a given environment. If you do not select this option while provisioning Cloudera AI Workbenches, then integration with Atlas will not work.

About this task

Procedure

1. Go to Cloudera AI and click Provision Workbench on the top-right corner.
2. Enter the Cloudera AI Workbench name and other details.
3. Click Advanced Options.
4. Select Enable Governance.

Registering training data lineage using a linking file

The Cloudera AI projects, model builds, model deployments, and associated metadata are tracked in Apache Atlas, which is available in the environment's SDX cluster. You can also specify additional metadata to be tracked for a given model build. For example, you can specify metadata that links training data to a project through a special file called the linking file (lineage.yaml).

The lineage.yaml file describes additional metadata and the lineage relationships between the project’s models and training data. You can use a single lineage.yaml file for all the models within the project.

Note: Your lineage file should be present in your project before you create a model build. The lineage file is parsed and metadata is attached during the model build process.

1. Create a YAML file in your Cloudera AI project called lineage.yaml.

   If you have used a template to create your project, a lineage.yaml file should already exist in your project.
2. Insert statements in the file that describe the relationships you want to track between a model and the training data. You can include additional descriptive metadata through key-value pairs in a metadata section.

YAML	YAML Structure	Description
Model name	Top-level entry	A Cloudera AI model name associated with the current project. There can be more than one model per linking file.

YAML	YAML Structure	Description
hive_table_qualified_names	Second-level entry	This pre-defined key introduces sequence items that list the names of Hive tables used as training data.
Table names	Sequence items	The qualified names of Hive tables used as training data enclosed in double quotation marks. Qualified names are of the format *DB-NAME.TABLE-NAME@CLUSTER-NAME*
metadata	Second-level entry	This pre-defined key introduces additional metadata to be included in the Atlas representation of the relationship between the model and the training data.
KEY:*VALUE*	Third-level entries	Key-value pairs that describe information about how this data is used in the model. For example, consider including the query text that is used to extract training data or the name of the training file used.

The following example linking file shows entries for two models in your project: modelName1 and modelName2:

```
modelName1:                                 # the name of your model
  hive_table_qualified_names:               # this is a predefined key to link to
                                            # training data
    - "db.table1@namespace"                 # the qualifiedName of the hive_table
                                            # object representing training data
    - "db.table2@ns"
  metadata:                                 # this is a predefined key for
                                            # additional metadata
    key1: value1
    key2: value2
    query: "select id, name from table"     # suggested use case: query used to
                                            # extract training data
    training_file: "fit.py"                 # suggested use case: training file
                                            # used
modelName2:                                 # multiple models can be specified in
                                            # one file
  hive_table_qualified_names:
    - "db.table2@ns"
```

Viewing lineage for a model deployment in Atlas

You can view the lineage information for a particular model deployment and trace it back to the specific data that was used to train the model through the Atlas' Management Console.

Procedure

1. Navigate to Management Console Environments , select your environment, and then under Quick Links select Atlas.
2. Search for ml_model_deployment. Click the model deployment of your interest.
3. Click the Lineage tab to see a visualization of lineage information for the particular model deployment and trace it back to the specific data that was used to train the model.

   You can also search for a specific table, click through to its Lineage tab and see if the table has been used in any model deployments.

Model Metrics

Metrics are essential for tracking model performance. By using custom code, you can track specific model predictions and analyze the metrics.

Enabling model metrics

Metrics are used to track the performance of the models. When you enable model metrics while creating a workbench, the metrics are stored in a scalable metrics store. You can track individual model predictions and analyze metrics using custom code.

About this task

Procedure

1. Go to Cloudera AI and click Provision Workbench on the top-right corner.
2. Enter the Cloudera AI Workbench name and other details.
3. Click Advanced Options.
4. Select Enable Model Metrics.

   If you want to connect to an external (custom) Postgres database, then specify the details in the additional optional arguments that are displayed. If you do not specify these details, a managed Postgres database will be used to store the metrics.

Tracking model metrics without deploying a model

Cloudera recommends that you develop and test model metrics in a workbench session before actually deploying the model. This workflow avoids the need to rebuild and redeploy a model to test every change.

Metrics tracked in this way are stored in a local, in-memory datastore instead of the metrics database, and are no longer stored when the session exits. You can access these metrics in the same session using the regular metrics API in the cdsw.py file.

The following example demonstrates how to track metrics locally within a session, and use the read_metrics function to read the metrics in the same session by querying by the time window.

To try this feature in the local development mode, use the following files from the Python template project:

- use_model_metrics.py
- predict_with_metrics.py

The predict function from the predict_with_metrics.py file shown in the following example is similar to the function with the same name in the predict.py file. It takes input and returns output, and can be deployed as a model. But unlike the function in the predict.py file, the predict function from the predict_with_metrics.py file tracks mathematical metrics. These metrics can include information such as input, output, feature values, convergence metrics, and error estimates. In this simple example, only input and output are tracked. The function is equipped to track metrics by applying the decorator models.cml_model(metrics=True).

```
import pickle
import cml.metrics_v1 as metrics
import cml.models_v1 as models

model = pickle.load(open('model.pkl', 'rb'))
# The model_metrics decorator equips the predict function to
# call track_metrics. It also changes the return type. If the
# raw predict function returns a value "result", the wrapped
# function will return eg
# {
#   "uuid": "612a0f17-33ad-4c41-8944-df15183ac5bd",
#   "prediction": "result"
# }
# The UUID can be used to query the stored metrics for this
# prediction later.
@models.cml_model(metrics=True)
```

```
def predict(args):
  # Track the input.
  metrics.track_metric("input", args)
  # If this model involved features, ie transformations of the
  # raw input, they could be tracked as well.
  # cdsw.track_metric("feature_vars", {"a":1,"b":23})
  petal_length = float(args.get('petal_length'))
  result = model.predict([[petal_length]])

  # Track the output.
  metrics.track_metric("predict_result", result[0][0])
  return result[0][0]
```

You can fetch the metrics from the local, in-memory datastore by using the regular metrics API. To fetch the metrics, set the dev keyword argument to True in the use_model_metrics.py file. You can query the metrics by model, model build, or model deployment using the variables cdsw.dev_model_crn and cdsw.dev_model_build_crn or cdsw.dev _model_deploy_crn respectively.

For example:

```
end_timestamp_ms=int(round(time.time() * 1000))
cdsw.read_metrics(model_deployment_crn=cdsw.dev_model_deployment_crn,
start_timestamp_ms=0,
end_timestamp_ms=end_timestamp_ms,
dev=True)
```

where CRN denotes Cloudera Resource Name, which is a unique identifier from Cloudera, analogous to Amazon's ARN.

Tracking metrics for deployed models

When you have finished developing your metrics tracking code and the code that consumes the metrics, simply deploy the predict function from predict_with_metrics.py as a model. No code changes are necessary.

Calls to read_metrics, track_delayed_metrics, and track_aggregate_metrics need to be changed to take the CRN of the deployed model, build or deployment. These CRNs can be found in the model's **Overview** page.

Calls to the call_model function also requires the model's access key (model_access_key in use_model_metrics.py) from the model's **Settings** page. If authentication has been enabled for the model (the default), a model API key for the user (model_api_token in use_model_metrics.py) is also required. This can be obtained from the user's **Settings** page.

Using Registered Models

Registered Models offers a single view for models stored in Cloudera AI Registries across Cloudera Environments and facilitate easy deployment to Cloudera AI Inference service.

When you import models from Model Hub, the models are listed under Registered Models. Review all imported models and associated metadata, such as the model's associated environment, visibility, owner name, and created date.

This page lists all imported models and associated metadata, such as the model's associated environment, visibility, owner name, and created date. You can click on any model to view details about that model, and its versions, and deploy any specific version of the model to the Cloudera AI Inference service. You can also delete a specific version of the model on this page. When you try to import a model with the same name, a new version of that model is added which can then be viewed under Registered Models.

Deploying a model from Registered Models

You can deploy a model from the Registered Models page into Cloudera AI Inference service.

Procedure

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

   The **Cloudera AI Workbenches** page displays.
2. Click **Registered Models** under **Deployments** in the left navigation menu.

   The **Registered Models** page displays. The page lists all the models of different Cloudera AI Registries along with the associated metadata.
3. Select the model you want to deploy. The **Registered Models** page displays the model information and the available versions of the model.
4. Click Deploy to deploy the latest version.

   You can select All from the version drop-down to view all the versions and click in the respective row of that version. You can deploy any version of the model the status of which is displayed as Ready. The Deploy Model dialog box is displayed.
5. In the Deploy Model dialog box, select the cluster of the Cloudera AI Inference service in which you want to deploy this model. The Create Endpoint page is displayed.
6. Enter the information and click OK.

   For information on creating an endpoint, see *Using Cloudera AI Inference service*.

Related Information

Using Cloudera AI Inference service

Viewing details of a registered model

You can view details like version information about the models in your AI Registries in the Registered Models page. By default, the latest version information is displayed. Model card describes the model, governing terms, the family of models, resources used, and further information on how to use the model, and so on. It also provides information about various versions, optimizations made in the specific version, and the minimum resource configuration required to deploy those versions.

Procedure

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

   The **Cloudera AI Workbenches** page displays.
2. Click **Registered Models** under **Deployments** in the left navigation menu.

   The **Registered Models** page displays. You can see all the registered models, associated environment, their owner, visibility, and the last updated time.
3. You can use the filter bar at the top of the window to filter the list of registered models by tag and environment name.
4. Select a registered model to see its description.

Editing model visibility

You can modify the visibility of the model to private or public status. If the visibility is set to Public, the registered model is available for all the users irrespective of their role. If the visibility is set to Private, the model is available only to the owner and the administrators of that environment.

Procedure

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

   The **Cloudera AI Workbenches** page displays.
2. Click **Registered Models** under **Deployments** in the left navigation menu.

   The **Registered Models** page displays.
3. Select the model whose visibility you want to change.
4. Click Edit Model and change the visibility of the model to Private or Public and the description of the model, if needed.
5. Click Update.

Deleting a registered model version

If you no longer want to access a version of a registered model, you can delete it.

Procedure

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

   The **Cloudera AI Workbenches** page displays.
2. Click **Registered Models** under **Deployments** in the left navigation menu.

   The **Registered Models** page displays.
3. Select the model you want to delete .
4. Click All from the version drop-down menu to view all the versions.
5. From the Actions menu, click .
6. Click OK to confirm the deletion.