H2O.ai

H2O MLOps

DOCUMENTATION

For questions, please contact **support@h2o.ai**

Contents

What is H2O MLOps?

H2O MLOps is an open, interoperable platform for model deployment, management, governance, monitoring, and alerting that features integration with H2O Driverless AI, H2O-3 open source, H2O Hydrogen Torch and third-party models.

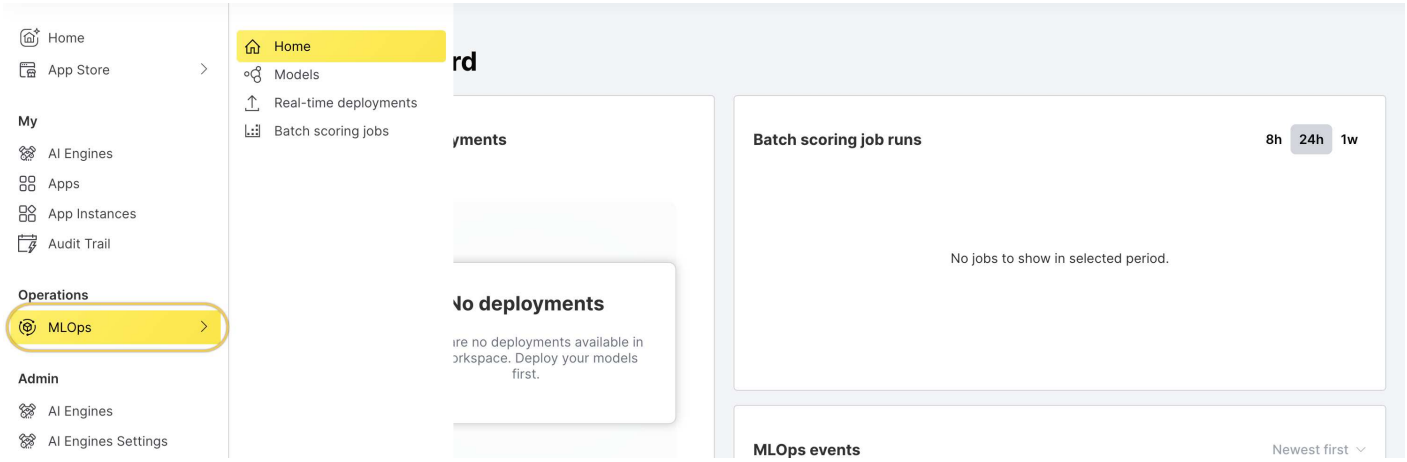
You can access H2O MLOps through:

- An interactive user interface (UI) available on the H2O AI Cloud.
- A Python client, which allows you to perform the same tasks from a Python application.

The H2O MLOps Python client can be installed from PyPI.

Access H2O MLOps

To access H2O MLOps, in the left navigation panel of H2O AI Cloud, click **MLOps** under **Operations**.



Install the H2O MLOps Python client

For instructions on installing the H2O MLOps Python client, see Install the H2O MLOps Python client.

Workflow

Overview

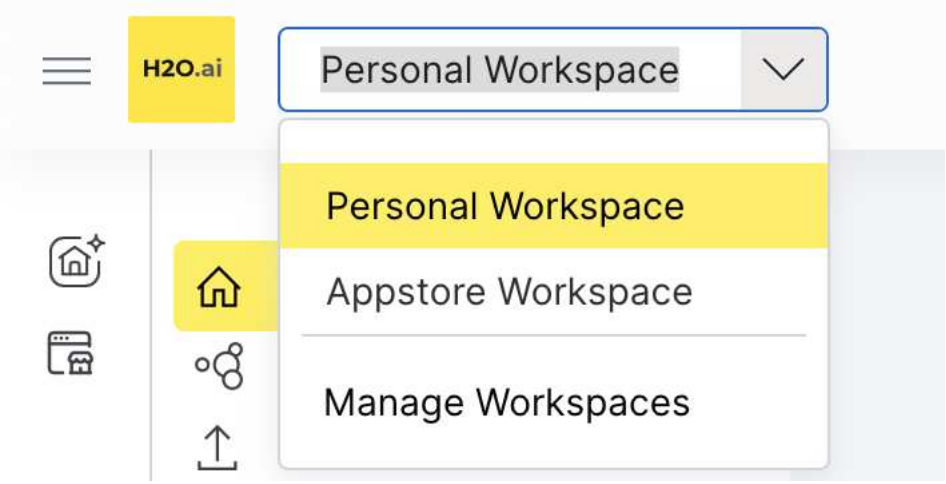
The typical H2O MLOps workflow can be summarized in the following sequential steps:

- Step 1: Select the workspace
- Step 2: Add a model
- Step 3: Deploy the model version
- Step 4: Score against the deployment
- Step 5: Monitor the deployed model

In the below sections, each step above, in turn, is summarized.

Step 1: Select the workspace

To begin, select your workspace from the workspaces drop-down in the top navigation bar.

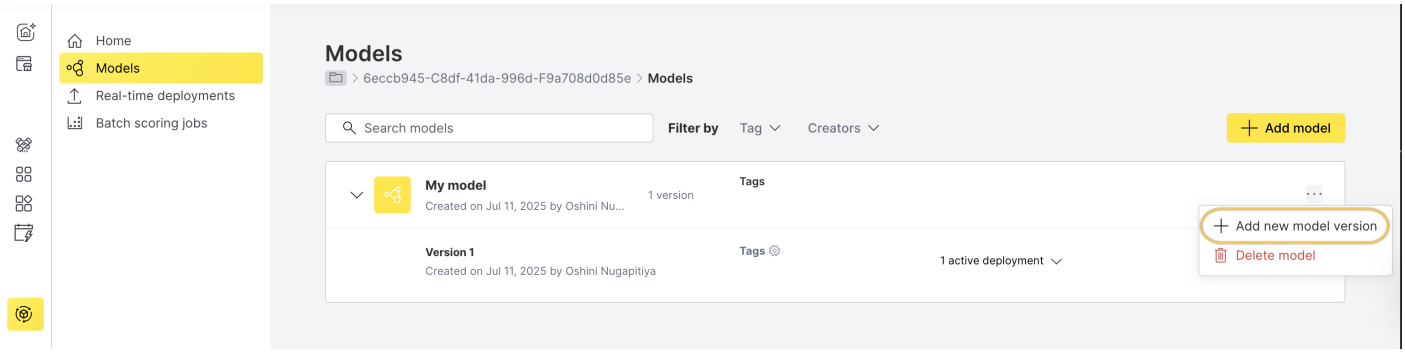


- To learn how to create and manage a workspace, see Create and manage a workspace.
- To learn how to manage workspaces using H2O MLOps Python client, see Manage workspaces.

Step 2: Add a model

After selecting a workspace, go to the **Models** tab in H2O MLOps and add your machine learning model.

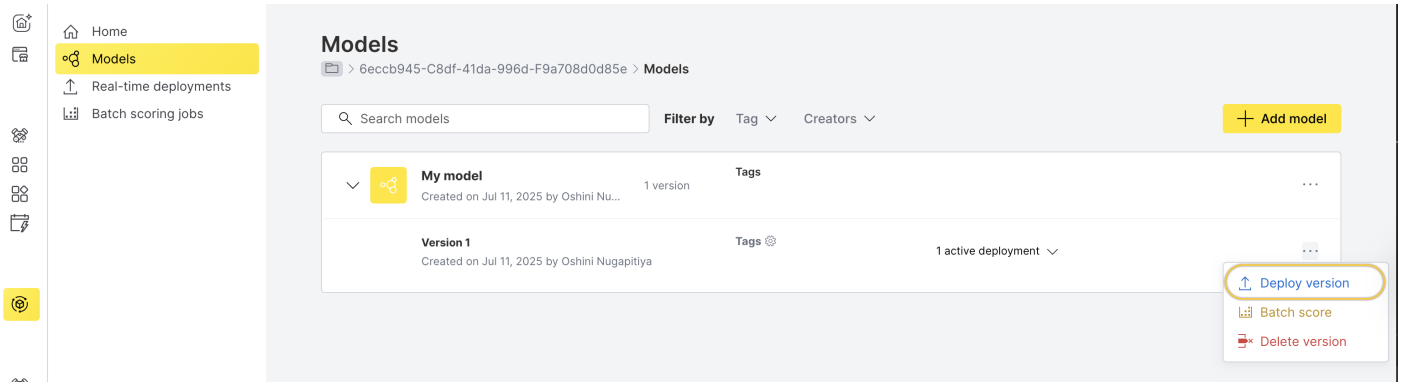
The first model you add becomes the initial version. You can add more versions later.



- To learn more about models, see Understand models.
- To learn how to add a model, see Add models.
- To learn how to view models, see View models.
- To learn how to manage models using H2O MLOps Python client, see Manage models.

Step 3: Deploy the model version

After adding a model, create a deployment for a model version so you can score and monitor its performance.



- To learn about deployments, see Understand deployments in MLOps.
- To learn how to create a deployment, see Create a deployment.
- To learn how to view deployments, see View deployments.
- To learn how to configure a deployment using H2O MLOps Python client, see Configure deployments.
- To learn how to manage deployments using H2O MLOps Python client, see Manage deployments.

Step 4: Score against the deployment

After deploying the model, you can run quick scoring on the deployment.

- To learn more about quick scoring, see Quick scoring.
- To learn how to score against deployments using H2O MLOps Python client, see Deployment scorer.

Step 5: Monitor the deployed model

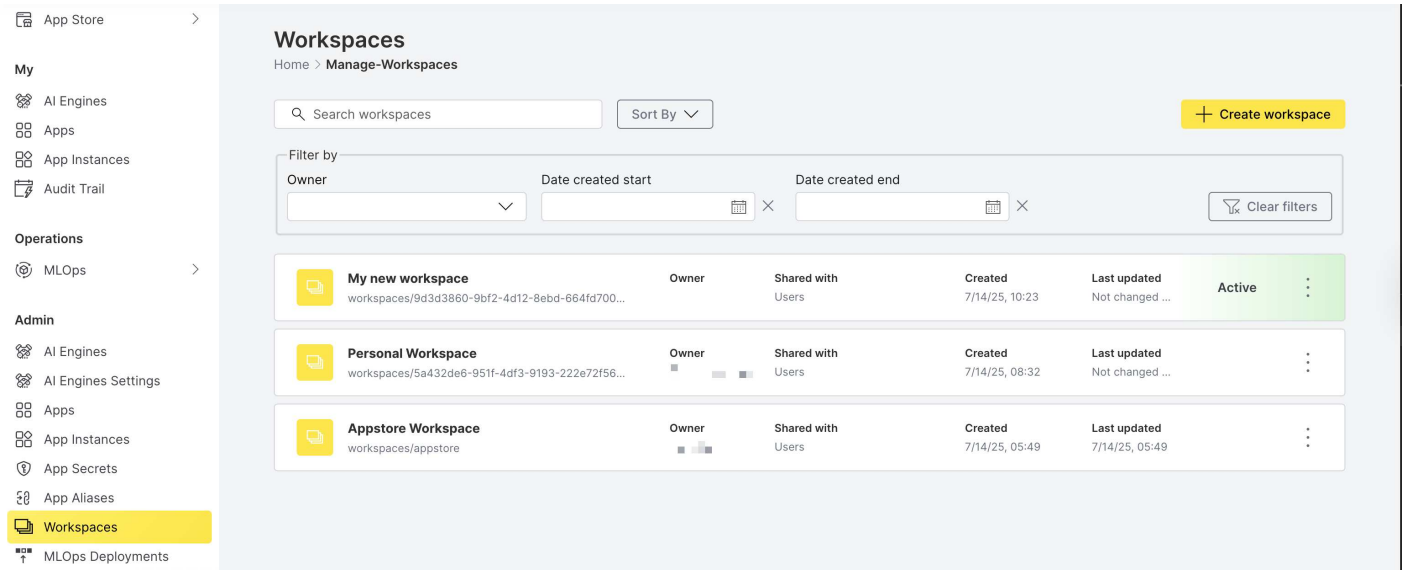
After scoring, monitor the deployed model to track its performance and detect issues such as model drift.

Note: You must enable and configure model monitoring when you create the deployment.

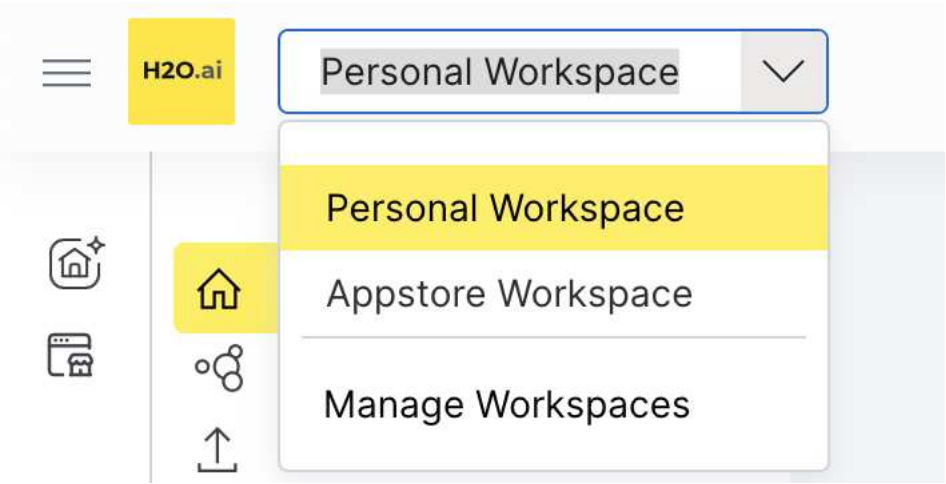
- To learn more about model monitoring, see Model monitoring.
- To learn how to configure monitoring for your deployment using the H2O MLOps Python client, see Monitoring setup.

Workspaces

A workspace is a centralized, dedicated environment within the H2O AI Cloud platform that enables you to share resources and collaborate on machine learning operations within the same H2O MLOps instance. Workspaces support role-based resource sharing while allowing fine-grained access control over those shared resources.



Before you start working in H2O MLOps, you must select a workspace. Use the workspace dropdown menu in the top navigation bar to make your selection. All operations performed within H2O MLOps are scoped to the selected workspace.



Note: Any project created in H2O Driverless AI is automatically synchronized with the H2O AI Cloud platform and creates a new workspace with the same name, which you can use in H2O MLOps.

To learn more about creating and managing workspaces, see Create and manage a workspace.

To learn how to create and manage workspaces using the H2O MLOps Python client, see Manage workspaces.

Understand models

H2O MLOps lets you register individual experiments and group them as versions of a registered model to organize a collection of experiments efficiently.

Before deploying an experiment, you must register it in H2O MLOps. You can either register it as a new model or add it as a new version under an existing model. Choosing the second option creates a new version of the existing model.

A **registered model** is a collection of individual model versions. Registered models group related model versions that solve a specific problem. You can register new experiments and iterations as updated versions of the model.

A **model version** has a one-to-one relationship with an experiment within a given Workspace. When you're ready to serve your best experiment, register it as a model version.

Note:

- When you register an experiment as a model, all data and metadata lineage are maintained.
- Model versions can be served in multiple deployments. There is no limitation on the number of deployments a single model version can be a part of.
- In any given workspace, an experiment can only be registered as one model version. This allows for a one-to-one mapping between an experiment and the model version.

Model schema

Each model can be described by its input and output column names and their types. Knowing the model schema is essential for monitoring purposes. Currently, only models using the known schema can be deployed by MLOps.

Model schema is represented by the experiment metadata attached to the experiment. Deployer expects the model schema to be stored in the `json_value` of the `input_schema` and `output_schema` keys.

Note: Natively supported Driverless AI MOJO2 and H2O-3 MOJO2 models are not required to contain the schema as the schema is an integral part of the MOJO2 artifacts.

Schema format

The following example shows how model schema is formatted:

```
[{"name": "<column name>", "type": "<column type>"}, ... ]
```

Column types

The following is a list of supported column types:

- Boolean
- Time64
- Float32
- Float64
- Int32
- Int64
- String

Note: Column type names are not case-sensitive.

Model type

Model types enumerate types outputted by artifact processors. This indirection is included due to the fact that one particular artifact type can contain multiple internal artifacts, each of which may be consumed by different runtimes. One particular artifact type can be processed in different ways, producing different outputs consumable by different runtimes.

A model type defines what runtime can be used for artifact deployment.

To learn more about models in H2O MLOps, see:

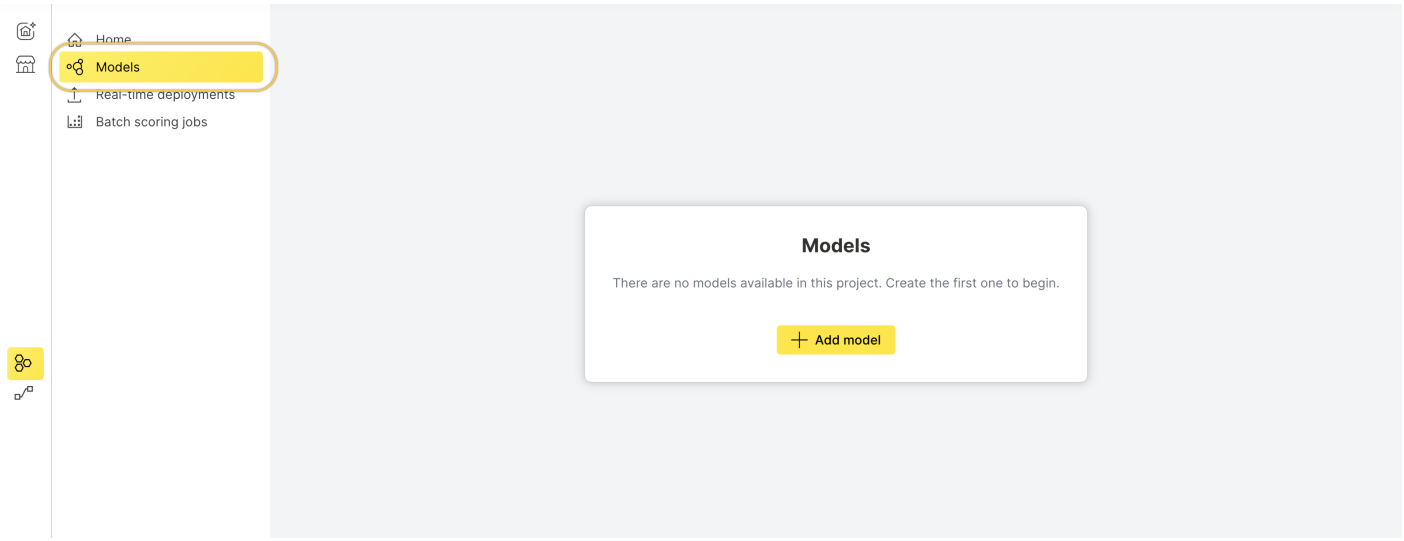
- Add models
- View models

- MLOps model support

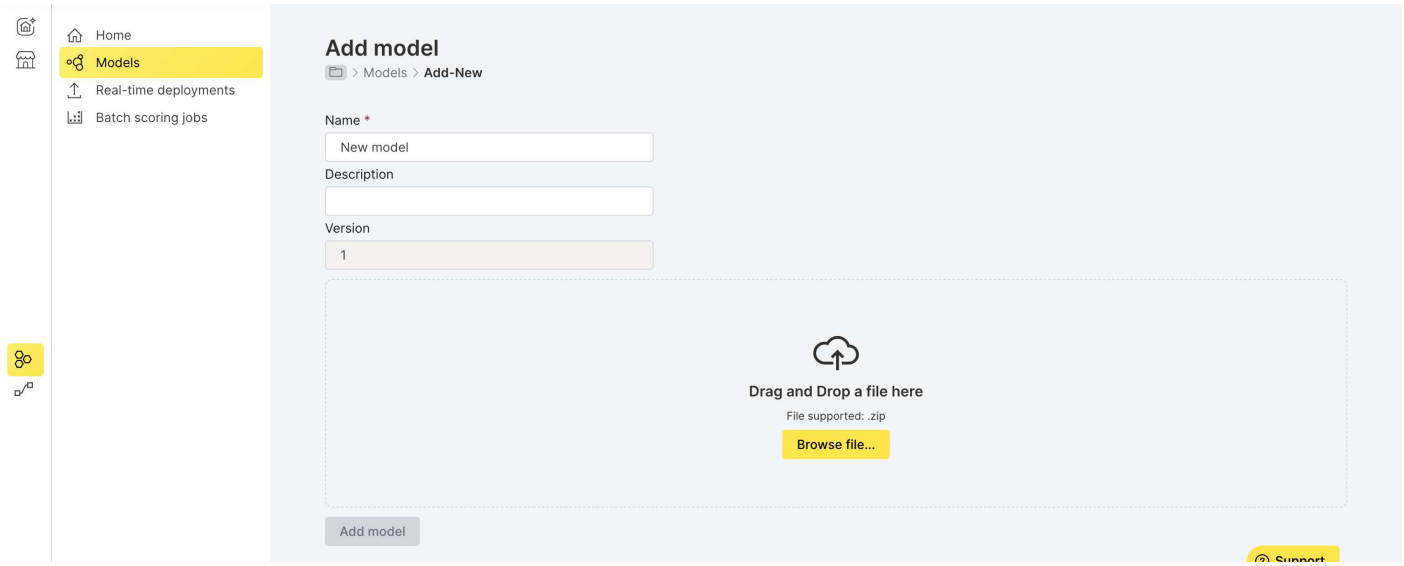
Add models

The following steps show how to add a model to an H2O MLOps workspace.

1. Click **Models** in the left navigation panel.



2. Click **Add model**. 3. In the **Name** text box, enter a name for the model. 4. In the **Description** text box, enter a description for the model.



5. Click Browse files, and upload the model file. **Note:** Only ZIP files are supported.

6. Click **Add model**.

View models

To view the list of models in your workspace, click **Models** in the left navigation panel.

To view details for a specific model, click the model name. This action opens the **Model details** panel.

Understand the Model details panel

The model details panel displays the model versions, parameters, metadata, and actions related to a specific model.

Update the model name and description

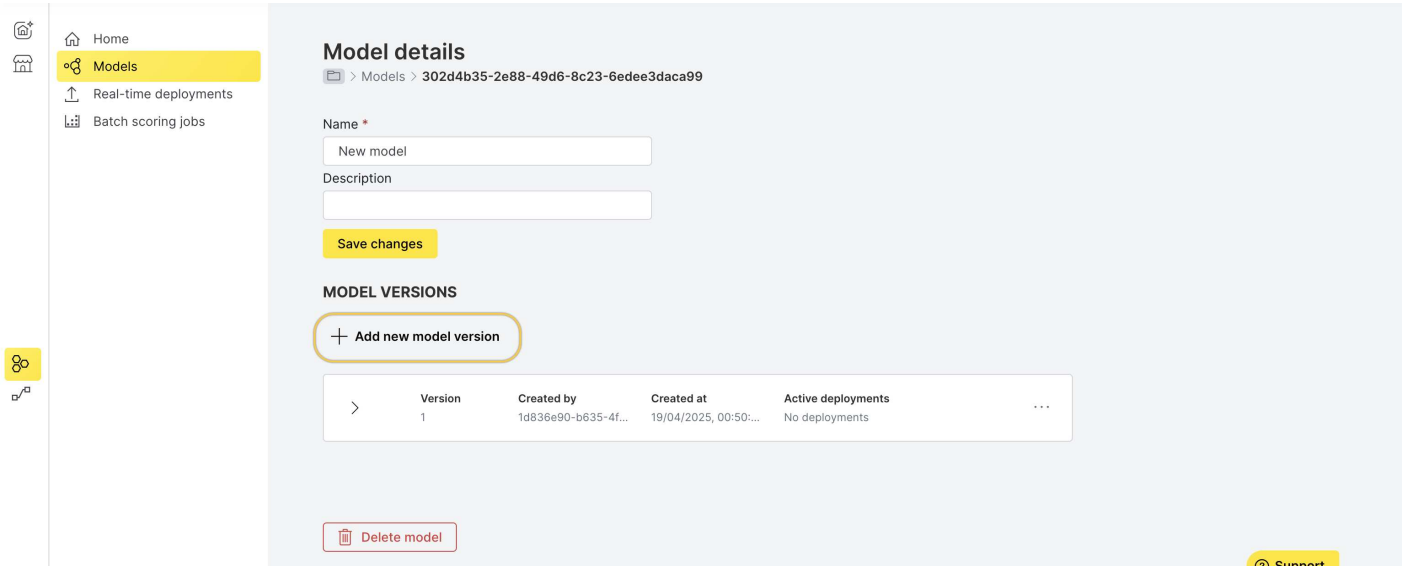
You can edit the model name and description directly in the panel.

1. Enter a new name for the model.
2. Enter a new description.
3. Click **Save changes**.

Add a new model version

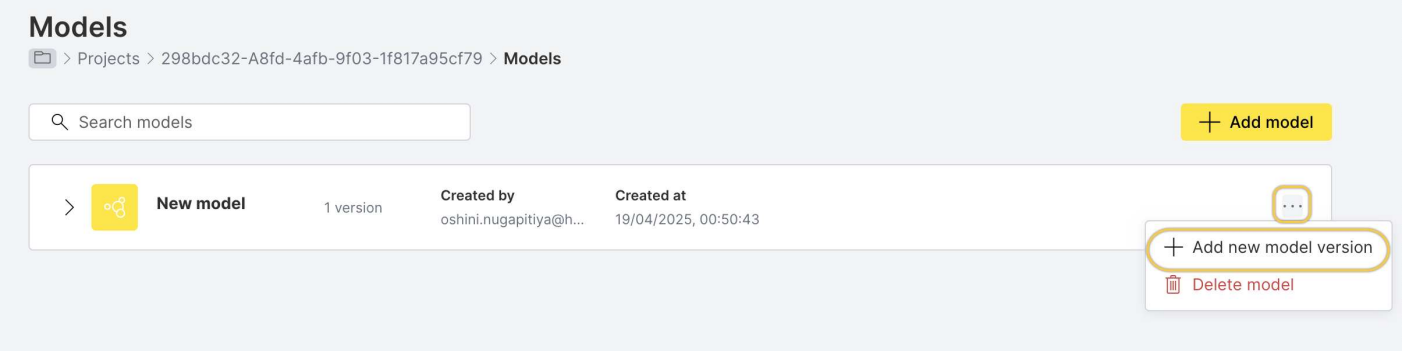
To add a new version of the model:

1. In the **Model versions** section, click **Add new model version**.



2. Click **Browse file** and upload the new model version. 3. Click **Add new version**.

Alternatively, you can click the menu icon for a specific model in the **Models** page and select **Add new model version**.



View model versions

In the **Model versions** section, expand a version to view the following details:

- **Version:** The version number of the model

- **Created by:** The ID of the user who created the version
- **Created at:** The timestamp when the version was created
- **Tags:** Tags to identify and group related model versions together. To add a tag, click **Tags**, then click **Manage tags** > **New tag**, enter the tag name, and click **Create tag**.
- **Active deployments:** The number of active deployments using this version
- **Comments:** Information to share with collaborators. Enter a comment and click **Add comment**.
- **Parameters:** Model-specific parameters
- **Metadata:** Additional information related to the model

MLOps model support

In H2O MLOps, the following model formats are supported:

- H2O Driverless AI MOJO pipeline / Python scoring pipeline
- H2O-3 open-source MOJO
- H2O Hydrogen Torch MLflow artifact
- Third-party model frameworks through MLflow

Note: Before a model can be deployed, it must first be registered in the H2O MLOps Model Registry.

H2O Driverless AI MOJO pipeline / Python scoring pipeline

You can import H2O Driverless AI MOJO / scoring pipeline `.zip` files directly through the H2O DAI interface or by dragging and dropping the file.

Alternatively, you can create an H2O MLOps deployment directly from the completed experiment page in H2O Driverless AI.

For more information, see Deploy Driverless AI models.

The following H2O Driverless AI versions are compatible with H2O MLOps v1.0.0:

- 1.10.7
- 1.10.7.1
- 1.10.7.2
- 1.10.7.3
- 1.10.7.4
- 1.10.7.5
- 1.11.0
- 1.11.1.1
- 2.0.0
- 2.1.0
- 2.2.0
- 2.2.1
- 2.2.2

H2O-3 open-source MOJO

You can upload an H2O-3 open-source MOJO .zip file by dragging and dropping it into the interface to deploy it in H2O MLOps.

H2O Hydrogen Torch MLflow artifact

After building a model in H2O Hydrogen Torch, you can deploy it to H2O MLOps. To learn more, see Deploy a model to H2O MLOps through the H2O Hydrogen Torch UI.

You can also download an H2O MLOps scoring pipeline for a model built in H2O Hydrogen Torch and use it to score new data through the H2O MLOps REST API. To learn more, see H2O MLOps pipeline.

Third-party model frameworks through MLflow

Third-party model frameworks include scikit-learn, PyTorch, XGBoost, LightGBM, and TensorFlow. You can import models by dragging and dropping an MLflow-packaged file.

To learn more about adding third-party models to H2O MLOps, see MLflow model support.

The following examples demonstrate how to wrap third-party model frameworks such as PyTorch, scikit-learn, and custom Python models using MLflow. You can then upload the resulting MLflow `.zip` file directly to H2O MLOps.

Example walkthrough

Note: Before you begin

- Install MLflow

- Install PyTorch
- Install scikit-learn

MLflow PyTorch

```
# Train PyTorch model
X_train, y_train = sklearn.datasets.load_wine(return_X_y=True, as_frame=True)
X_tensor = torch.from_numpy(X_train.to_numpy())
y_tensor = torch.from_numpy(y_train.to_numpy())
dataset = torch.utils.data.dataset.TensorDataset(X_tensor, y_tensor)

torch_model = torch.nn.Linear(13, 1)
loss_fn = torch.nn.MSELoss(reduction="sum")

learning_rate = 1e-6
for batch in dataset:
torch_model.zero_grad()

X, y = batch
y_prediction = torch_model(X.float())
loss = loss_fn(y_prediction, y.float())
loss.backward()

with torch.no_grad():
for param in torch_model.parameters():
param -= learning_rate * param.grad

# Infering and setting model signature
# Model signature is mandatory for models that are going to be loadable by the server.
# Only ColSpec inputs and output are supported.
model_signature = signature.infer_signature(X_train)
model_signature.outputs = mlflow.types.Schema(
[mlflow.types.ColSpec(name="quality", type=mlflow.types.DataType.float)]
)

# Save as MLflow and zip the artifact.
model_tmp = tempfile.TemporaryDirectory()
try:
model_dir_path = os.path.join(model_tmp.name, "wine_model")
mlflow.pytorch.save_model(
torch_model, model_dir_path, signature=model_signature
)
zip_path = shutil.make_archive(
os.path.join(model_tmp.name, "artifact"), "zip", model_dir_path
)
final_zip_path = os.path.abspath("wine_model_artifact.zip")
shutil.copy(zip_path, final_zip_path)
finally:
model_tmp.cleanup()
```

MLflow scikit-learn

```
# Train the scikit-learn model.
X_train, y_train = sklearn.datasets.load_wine(return_X_y=True, as_frame=True)
y_train = (y_train >= 7).astype(int)

sklearn_model = ensemble.RandomForestClassifier(n_estimators=10)
sklearn_model.fit(X_train, y_train)

# Infering and setting model signature
# Model signature is mandatory for models that are going to be loadable by the server.
```

```
# Only ColSpec inputs and output are supported.
model_signature = signature.infer_signature(X_train)
model_signature.outputs = mlflow.types.Schema(
[mlflow.types.ColSpec(name="quality", type=mlflow.types.DataType.float)]
)

# Save as MLflow and zip the artifact.
model_tmp = tempfile.TemporaryDirectory()
try:
model_dir_path = os.path.join(model_tmp.name, "wine_model")
mlflow.sklearn.save_model(
sklearn_model, model_dir_path, signature=model_signature
)
zip_path = shutil.make_archive(
os.path.join(model_tmp.name, "artifact"), "zip", model_dir_path
)
final_zip_path = os.path.abspath("wine_model_artifact.zip")
shutil.copy(zip_path, final_zip_path)
finally:
model_tmp.cleanup()
```

MLflow custom Python model

```
### Include the custom model wrapper.
class RandomForestWithVectorizor(mlflow.pyfunc.PythonModel):

def load_context(self, context):
import pickle
with open(context.artifacts["vectorizor"], "rb") as f:
self.vectorizor = pickle.load(f)
with open(context.artifacts["svd"], "rb") as f:
self.svd = pickle.load(f)
with open(context.artifacts["rf"], "rb") as f:
self.rf = pickle.load(f)

def predict(self, context, model_input):
input_vec_tfidf = self.vectorizor.transform(
self.get_input_column(model_input)
)
input_vec = self.svd.transform(input_vec_tfidf)
return self.rf.predict(input_vec)

def get_input_column(self, model_input):
return model_input["Description"]

### Train/Fit the necessary components for the model.
# TfidfVectorizer, TruncatedSVD, RandomForestClassifier
data_url = "https://h2o-public-test-data.s3.amazonaws.com/smalldata/amazon-food-review/"
train_data = pd.read_csv(f"{data_url}/AmazonFineFoodReviews-train-26k.csv")

# Fit data transformers: TfidfVectorizer and TruncatedSVD
vectorizor = text.TfidfVectorizer(stop_words="english")
train_tfidf_vector = vectorizor.fit_transform(train_data["Description"])

svd = decomposition.TruncatedSVD(n_components=300)
train_vector = svd.fit_transform(train_tfidf_vector)

# Train RandomForestClassifier that consumes a vector
rf = ensemble.RandomForestClassifier(n_estimators=50)
rf.fit(train_vector, train_data["PositiveReview"])
```

```
### Create and set the model signature.
input_schema = mlflow.types.Schema([
mlflow.types.ColSpec(name="Description", type=mlflow.types.DataType.string)
])

output_schema = mlflow.types.Schema([
mlflow.types.ColSpec(name="PositiveReview", type=mlflow.types.DataType.integer)
])

model_signature = mlflow.models.signature.ModelSignature(
inputs=input_schema,
outputs=output_schema,
)

### Save as MLflow and zip the artifact.
model_tmp = tempfile.TemporaryDirectory()
try:
model_dir_path = os.path.join(model_tmp.name, "sentiment_model")
vectorizor_path = os.path.join(model_tmp.name, "vectorizor.pkl")
svd_path = os.path.join(model_tmp.name, "svd.pkl")
rf_path = os.path.join(model_tmp.name, "rf.pkl")

with open(vectorizor_path, "wb") as f:
pickle.dump(vectorizor, f)

with open(svd_path, "wb") as f:
pickle.dump(svd, f)

with open(rf_path, "wb") as f:
pickle.dump(rf, f)

# Create a dictionary to tell MLflow where the necessary artifacts are
artifacts = {
"vectorizor": vectorizor_path,
"svd": svd_path,
"rf": rf_path,
}
# Use above defined Custom Model Wrapper
mlflow.pyfunc.save_model(
path=model_dir_path,
python_model=RandomForestWithVectorizor(),
artifacts=artifacts,
signature=model_signature
)
zip_path = shutil.make_archive(
os.path.join(model_tmp.name, "artifact"), "zip", model_dir_path
)
final_zip_path = os.path.abspath("sentiment_model_artifact.zip")
shutil.copy(zip_path, final_zip_path)
finally:
model_tmp.cleanup()
```


MLflow model support

H2O MLOps lets you upload and deploy MLflow models. The following sections describe this feature.

Supported third-party models

The following is a list of tested and supported third-party Python models.

Package	Version
fastai	>=2.7.15,<2.9.0
keras	~=3.9.1
lightgbm	~=4.6.0
mlflow	>=2.14.2
onnx	~=1.17.0
scikit-learn	>=1.1.3,<1.4.0
statsmodels	~=0.14.1
tensorflow	~=2.16.1
torch	<2.6.0
torchvision	<0.21.0
xgboost	~=1.7.1
opencv-python-headless	~=4.11.0
click	<9.0.0,>=8.1.0

Create MLflow artifacts for third-party frameworks

The following is an example of how to create MLflow artifacts for third-party frameworks.

```
import shutil
import mlflow.sklearn
import sklearn.datasets
import sklearn.ensemble
from mlflow.models import signature
from mlflow.types import Schema, ColSpec, DataType

# Train sklearn model
X_train, y_train = sklearn.datasets.load_wine(return_X_y=True, as_frame=True)
y_train = (y_train >= 7).astype(int)

sklearn_model = sklearn.ensemble.RandomForestClassifier(n_estimators=10)
sklearn_model.fit(X_train, y_train)

# Infer and set model signature
model_signature = signature.infer_signature(X_train, sklearn_model.predict(X_train))
model_signature.outputs = Schema(
    [ColSpec(name="quality", type=DataType.float)]
)

# Define the path to store the model in the current directory
model_dir_path = "wine_model"

# Save the trained sklearn model with MLflow
mlflow.sklearn.save_model(
    sklearn_model, model_dir_path, signature=model_signature
)

# Create a zip archive of the saved model
shutil.make_archive("artifact", "zip", model_dir_path)
```

Understand deployments

In H2O MLOps, deployments are created when model version(s) are served for scoring. Model endpoint security, artifact type, runtime, and Kubernetes options can be configured when deploying a model.

H2O MLOps supports different deployment modes:

- **Real-time deployments:** Make a model available as a live REST endpoint that returns predictions immediately when given input data. Types of real-time deployments include:
  - **Single model deployments:** Serve one model version at a time.
  - **A/B test deployments:** Compare the performance of two or more models in production.
  - **Champion/Challenger deployments:** Continuously compare a Champion model against one or more Challenger models to promote the best performer.
- **Batch scoring deployments:** Run model scoring jobs on batches of data instead of serving predictions in real time.

You can create and manage deployments using:

- The H2O MLOps UI on H2O AI Cloud.
- The H2O MLOps Python client, which allows you to automate deployment tasks from a Python application.

To learn more about deployments, refer to the following pages:

- Create a deployment
- View deployments
- Scoring runtimes
- Vertical Pod Autoscaler (VPA) support
- Pod Disruption Budget (PDB)

To learn more about deployments using the Python client, see:

- Configure deployments
- Manage deployments
- Deployment scorer

Scoring runtimes

This page describes the scoring runtimes available for model deployment, including configuration options and usage instructions for each runtime type.

Runtime options

The selection of available runtimes is determined by the artifact type that you specify. The following list provides information on the available options when selecting an artifact type and runtime.

Caution: note Selecting an incorrect runtime causes the deployment to fail.

Artifact type	Runtime option(s)	Notes
Driverless AI MOJO pipeline/MLflow Driverless AI MOJO pipeline	`Driverless AI MOJO ScorerDriverless AI MOJO Scorer (Shapley original only)Driverless AI MOJO Scorer (Shapley transformed only)Driverless AI MOJO Scorer (Shapley all)Driverless AI MOJO Scorer (C++ Runtime)`	• **Original only** and **Transformed only** require **2× memory** vs. **Shapley none**. • **Shapley all** requires **3× memory**. • **C++ Runtime** requires experiment to be linked through workspace.
Driverless AI Python pipeline/MLflow Driverless AI Python pipeline	`Python Pipeline Scorer [Driverless AI 1.10.7--1.11.1.1, 2.0.0--2.2.3]`	Python scorer version must match the Driverless AI version used to build the model.
H2O-3 MOJO/MLflow H2O-3 MOJO	`H2O-3 MOJO ScorerH2O-3 MOJO Scorer (Shapley transformed only)`	
MLflow zipped	`[PY-3.10][CPU] HT Flexible Runtime[PY-3.10][GPU] HT Flexible Runtime[Py 3.9--3.12] Dynamic MLflow Model Scorer`	For usage details, see MLflow Dynamic Runtime.

Note:

- The C++ MOJO2 runtime (`Driverless AI MOJO Scorer (C++ Runtime)`) accepts a wider range of algorithms Driverless AI may use that the Java runtime does not support, including BERT, GrowNet, and TensorFlow models. If you want to use one of these models, it must be linked from Driverless AI and not be manually uploaded.
- MLflow runtimes support Python 3.9 and later.
- For end of support information on H2O Driverless AI runtimes, see the Driverless AI Prior Releases page.

Artifact names mapping

The following table describes the mapping of artifact names.

Artifact type name	Storage artifact type	Artifact type
DAI MOJO Pipeline	`dai/mojo_pipeline`	`dai_mojo_pipeline`
DAI Python Pipeline	`dai/scoring_pipeline`	`dai_python_scoring_pipeline`
H2O-3 MOJO	`h2o3/mojo`	`h2o3_mojo`
MLflow zipped	`python/mlflow`	`python/mlflow.zip`
MLflow DAI MOJO Pipeline	`mlflow/mojo_pipeline`	`mlflow_mojo_pipeline`
MLflow DAI Python Pipeline	`mlflow/scoring_pipeline`	`mlflow_scoring_pipeline`
MLflow H2O-3 MOJO	`mlflow/h2o3_mojo`	`mlflow_h2o3_mojo`

Runtime names mapping

The following table describes the mapping of runtime names.

Compatible model	Runtime name	Runtime
Driverless AI MOJO models (Java runtime)	Driverless AI MOJO Scorer	`dai_mojo_runtime`
Driverless AI MOJO models (C++ runtime) - supports all Shapley contribution types and is expected to have significantly lower memory usage	Driverless AI MOJO Scorer (C++ Runtime)	`dai-mojo-cpp_experimental`
Driverless AI MOJO models (Java runtime) - with Shapley contributions for original features	Driverless AI MOJO Scorer (Shapley original only)	`mojo_runtime_shapley_original`
Driverless AI MOJO models (Java runtime) - with Shapley contributions for transformed features	Driverless AI MOJO Scorer (Shapley transformed only)	`mojo_runtime_shapley_transformed`
Driverless AI MOJO models (Java runtime) - with Shapley contributions for both original and transformed features	Driverless AI MOJO Scorer (Shapley all)	`mojo_runtime_shapley_all`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.10.7	Python Pipeline Scorer [Driverless AI 1.10.7]	`python-scorer_dai_pipelines_1107`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.10.7.1	Python Pipeline Scorer [Driverless AI 1.10.7.1]	`python-scorer_dai_pipelines_11071`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.10.7.2	Python Pipeline Scorer [Driverless AI 1.10.7.2]	`python-scorer_dai_pipelines_11072`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.10.7.3	Python Pipeline Scorer [Driverless AI 1.10.7.3]	`python-scorer_dai_pipelines_11073`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.11.0	Python Pipeline Scorer [Driverless AI 1.11.0]	`python-scorer_dai_pipelines_1110`
Driverless AI Python Scoring Pipeline models created by Driverless AI 1.11.1.1	Python Pipeline Scorer [Driverless AI 1.11.1.1]	`python-scorer_dai_pipelines_11111`
Driverless AI Python Scoring Pipeline models created by Driverless AI 2.0.0	Python Pipeline Scorer [Driverless AI 2.0.0]	`python-scorer_dai_pipelines_200`
Driverless AI Python Scoring Pipeline models created by Driverless AI 2.1.0	Python Pipeline Scorer [Driverless AI 2.1.0]	`python-scorer_dai_pipelines_210`
Driverless AI Python Scoring Pipeline models created by Driverless AI 2.2.3	Python Pipeline Scorer [Driverless AI 2.2.3]	`python-scorer_dai_pipelines_223`
H2O-3 MOJO models	H2O-3 MOJO Scorer	`h2o3_mojo_runtime`
H2O-3 MOJO models	H2O-3 MOJO Scorer (Shapley transformed only)	`h2o3_mojo_runtime_shapley_transformed`
H2O Hydrogen Torch MLflow models	[PY-3.10][CPU] HT Flexible Runtime	`python-scorer_hydrogen_torch_cpu_py310`
H2O Hydrogen Torch MLflow models	[PY-3.10][GPU] HT Flexible Runtime	`python-scorer_hydrogen_torch_gpu_py310`
MLFlow non-H2O.ai models created with Python 3.9	[Py 3.9] Dynamic MLflow Model Scorer	`python-scorer_mlflow_dynamic_39`
MLFlow non-H2O.ai models created with Python 3.10	[Py 3.10] Dynamic MLflow Model Scorer	`python-scorer_mlflow_dynamic_310`
MLFlow non-H2O.ai models created with Python 3.11	[Py 3.11] Dynamic MLflow Model Scorer	`python-scorer_mlflow_dynamic_311`

Compatible model	Runtime name	Runtime
MLFlow non-H2O.ai models created with Python 3.12	[Py 3.12] Dynamic MLflow Model Scorer	`python-scorer_mlflow_dynamic_312`

MLflow Dynamic Runtime

The MLflow Dynamic Runtime lets you deploy MLflow models with diverse dependencies in H2O MLOps. The following steps describe how to deploy a dynamic MLflow runtime deployment in H2O MLOps.

Note: For an example of how to train a dynamic runtime, see Train a dynamic runtime.

1. Save your model using the `mlflow.pyfunc.save_model` function call. Use the `pip_requirements` parameter to specify the Python package dependencies required by the model.

```
mlflow.pyfunc.save_model(
   path=...,
   python_model=...,
   artifacts=...,
   signature=...,
   pip_requirements=..., # <- Use this parameter to override libs for dynamic runtime
)
```

2. After saving the model, create a zip archive of the saved model directory. Ensure that a requirements file (`requirements.txt`) that lists all dependencies is included in the zip archive. The following is an example of the expected structure for the zip file from a TensorFlow model:

```
tf-model-py310
   MLmodel
   artifacts
      tf.h5
   conda.yaml
   python_env.yaml
   python_model.pkl
   requirements.txt
```

3. Depending on whether you are using Python 3.9, Python 3.10, Python 3.11, or Python 3.12 select from one of the following options:

   - [PY-3.9] MLflow Dynamic Model Scorer
   - [PY-3.10] MLflow Dynamic Model Scorer
   - [PY-3.11] MLflow Dynamic Model Scorer
   - [PY-3.12] MLflow Dynamic Model Scorer

Note: The MLflow Dynamic Runtime has a fixed MLflow dependency, which is MLflow 2.14.2. This means that the MLflow Dynamic Runtime is not guaranteed to work with a different version of MLflow model.

Example: Train a dynamic runtime model

The following example demonstrates how to train a dynamic runtime with TensorFlow:

```
# Import libraries
import mlflow
import pandas as pd
import shutil
import tensorflow as tf
from sklearn import datasets

# Load and prepare data
diabetes = datasets.load_diabetes()
X = diabetes.data[:, 2:3]  # Use only one feature for simplicity
y = diabetes.target
```

```
# Build and train TensorFlow model
tf_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(1, input_dim=1)
])
tf_model.compile(optimizer='adam', loss='mean_squared_error')
tf_model.fit(X, y, epochs=10)

tf_model_path = "tf.h5"

tf_model.save(tf_model_path, save_format="h5")


# Enable the TensorFlow model to be used in the Pyfunc format
class PythonTFmodel(mlflow.pyfunc.PythonModel):
    def load_context(self, context):
        import tensorflow as tf
        self.model = tf.keras.models.load_model(context.artifacts["model"])

    def predict(self, context, model_input):
        tf_out = self.model.predict(model_input)
        return pd.DataFrame(tf_out, columns=["db_progress"])


# Generate signature from your model definition
model = PythonTFmodel()
context = mlflow.pyfunc.PythonModelContext(model_config=dict(), artifacts={"model": tf_model_path})
model.load_context(context)
x = pd.DataFrame(X, columns=["dense_input"])
y = model.predict(context, x)
signature = mlflow.models.signature.infer_signature(x, y)

# Specify a file path where the model will be saved
mlflow_model_path = "./tf-model-py310"

# Save model using MLflow
mlflow.pyfunc.save_model(
    path=mlflow_model_path,
    python_model=PythonTFmodel(),
    signature=signature,
    artifacts={"model": tf_model_path},
    pip_requirements=["tensorflow"]
)

# Package model as a zip archive
shutil.make_archive(
    mlflow_model_path, "zip", mlflow_model_path
)
```

The following is the structure of the zip file that is generated in the preceding example:

```
- tf-model-py310
    - MLmodel
    - artifacts
    - tf.h5
    - conda.yaml
    - python_env.yaml
    - python_model.pkl
    - requirements.txt
```

Generic Ephemeral volumes

The custom additional volumes feature now supports `emptyDir` volumes and `ephemeral` volumes.

Note: The `storageClassName` property for volumes is optional. If not provided, the default storage class will be used.

Example configuration

```
# Custom additional volumes with selected mount paths.
# This section, as well as each of its fields, is optional.
volume-mounts = [
  {
    name = "ephemeral_volume"
    type = "ephemeral"
    properties = [
      { name = "size", value = "1Gi" }
    ]
    paths = ["/ephemeral_volume_1", "/ephemeral_volume_2"]
  },
  {
    name = "emptyDir_volume"
    type = "emptyDir"
    properties = [
      { name = "medium", value = "Memory" }
    ]
    paths = ["/emptyDir_volume_1", "/emptyDir_volume_2"]
  }
]
```

YAML configuration The `volumeMounts` section should be added to the runtime specification of the Helm Chart.

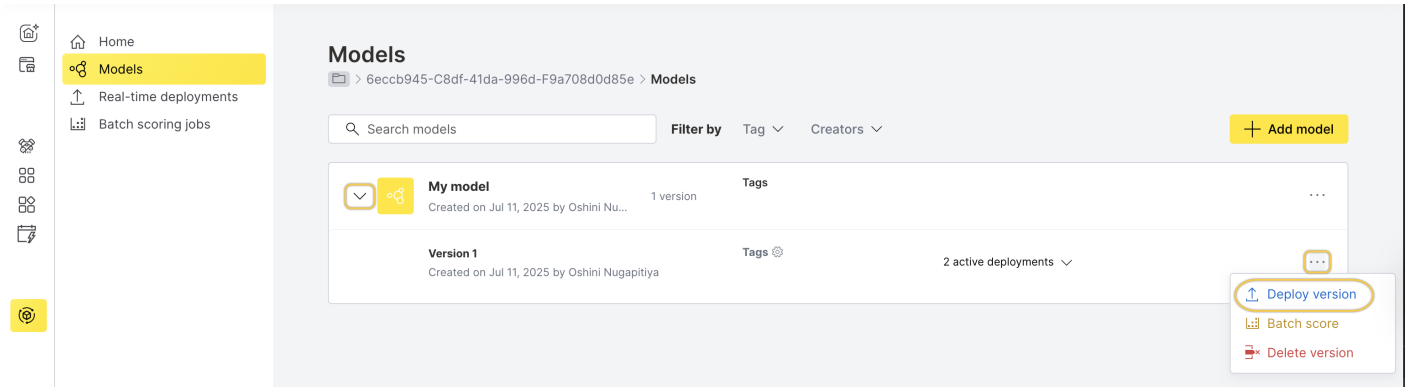
```
runtimes:
    volumeMounts:

        - name: "dev-shm"
        type: "ephemeral"
        properties:
            size: "1Gi"
        paths: ["/tmp"]
```

Create a deployment

Note: You must add model versions to H2O MLOps before deploying them. For more information, see Add models.

Follow these steps to deploy a model version in H2O MLOps:

1. In the left navigation panel, click **Models**.
2. Expand the model that includes the version you want to deploy.
3. Click the **More options** menu ( ) next to the version, then select **Deploy version**.

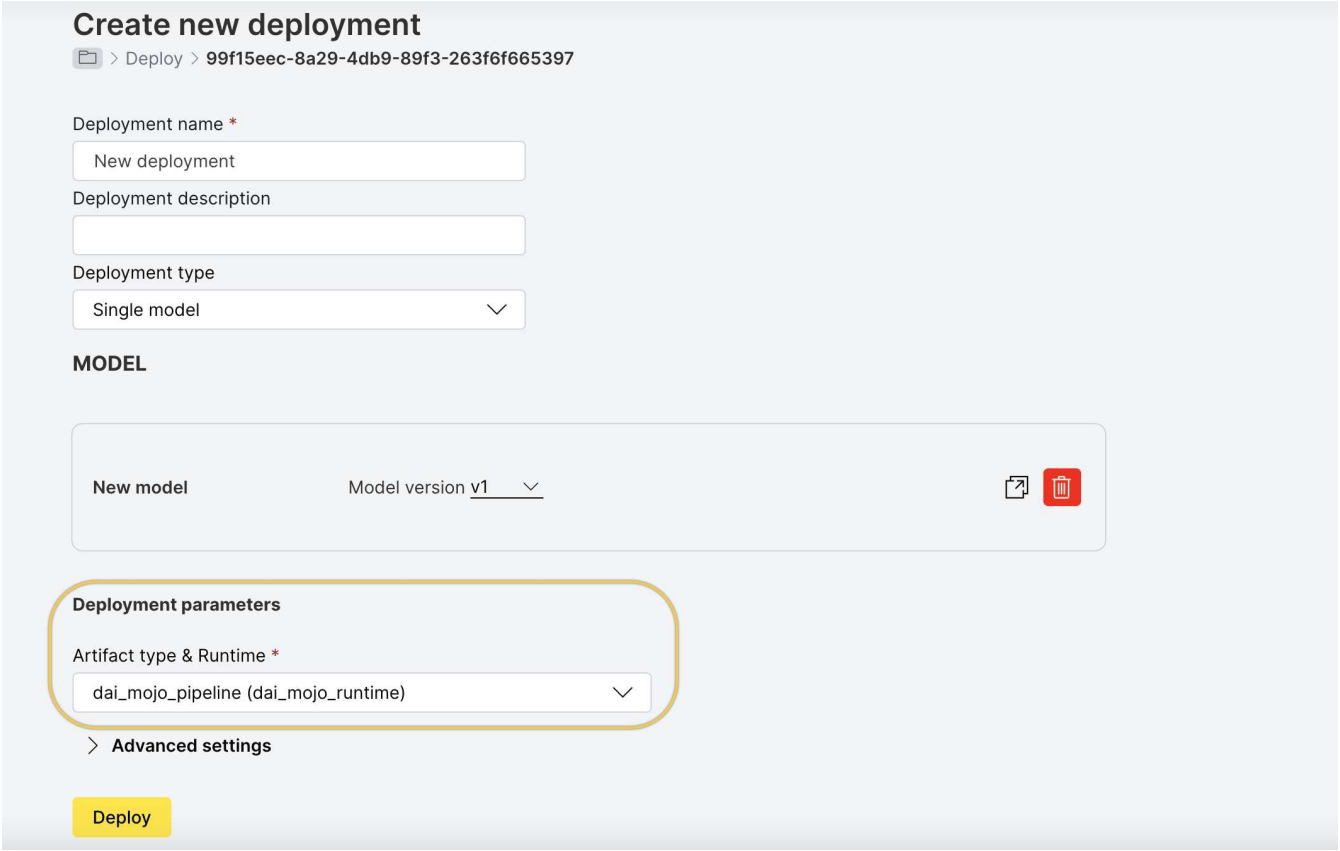


4. In the **Create new deployment** page, specify the following:
   1. Enter a name and, optionally, a description for the deployment.
   2.

```
Specify a deployment type for the deployment. Select one of the following options:

- **Single model:** Select one model/model version for the deployment.

![](../assets/single-model-selection.png)

- **A/B test:** A/B testing in MLOps lets you compare the performance of two or more models. When requests ar

![](../assets/ab-test-model-selection.png)

- **Champion/challenger:** This deployment mode lets you continuously compare your chosen best model (Champio

![](../assets/champion-challenger-model-selection.png)
```

Note: For A/B test and champion/challenger deployments, use two models with the same schema. That is, the models must have identical input and output column names and data types.

3. **Artifact type and runtime:** Select an artifact type and runtime for the deployment. (**Note:** For more information on scoring runtimes in H2O MLOps, see Scoring runtimes.) Each entry in the Artifact Type and Runtime drop-down represents an artifact that is linked to the selected model and is deployable by MLOps. Note that admins have the option to configure and add additional runtimes.



4. Click **Deploy**.

Advanced settings

This section describes various advanced settings that you can configure for your deployment. To expand the Advanced settings section, click the > expander arrow in the **Create new deployment** page.

Kubernetes options

Each of the following Kubernetes (K8s) options (resource requests, limits) are applied on a per-replica basis.

Replicas (optional) Specify the number of static replicas of the model you want to deploy. This is the number of concurrent pods (instances of the model) that are deployed. Deploying multiple replicas enables the model to handle more simultaneous scoring requests.

Specifying multiple replicas is useful for load balancing and achieving high availability for scoring tasks. For H2O MLOps version 0.65 and later, set replicas to zero to scale down the deployment. In versions prior to 0.65, set the number of replicas to -1 to achieve the same effect.

Note: Each of the following Kubernetes options (resource requests, limits) are applied on a per-replica basis.

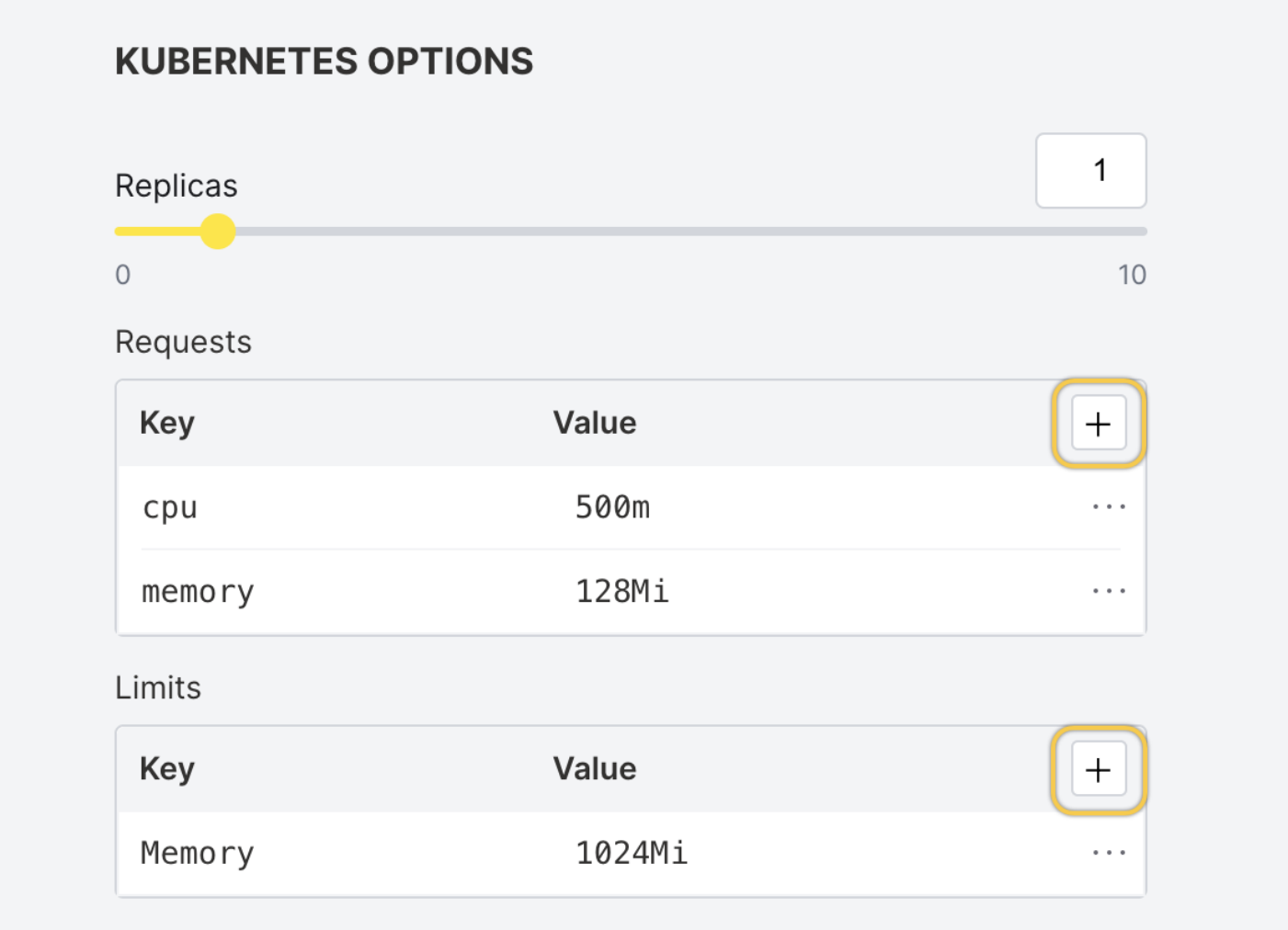
Resource requests and limits (optional) Use resource requests to specify the expected memory or CPU usage for a model. Resource limits define the maximum usage allowed. This helps Kubernetes manage scheduling and prevent resource overuse.

- **Requests:** Resource *requests* determine the amount of resources the deployment asks Kubernetes to provide it. For example, if a replica requests 256Mi of memory, Kubernetes schedules the deployment on any node that has enough memory available to satisfy the stated requirement.

- **Limits:** Resource *limits* determine the maximum amount of resources that are available to a deployment. If a deployment exceeds a limit on the amount of memory allocated to it, the deployment is restarted. Note that the deployment does not restart if it exceeds a CPU limit.

Note: By default, the Kubernetes resource requests and limits fields are automatically populated based on the selected runtime and artifact type.

Additional fields for any existing custom resources that have been set up by an admin in your Kubernetes cluster can be added by clicking the + button.



Note:

- The resource requests and limits fields must be defined using the quantity suffixes used in Kubernetes. The default values for Memory and CPU requests are 256Mi and 100m respectively. For more information, see Resource units in Kubernetes.
- By default, resources are not limited.
- When specifying custom resources, if Kubernetes is not able to satisfy the stated custom value(s), then the pod cannot be scheduled.
- You can use the H2O MLOps Python client to update the Kubernetes resources and replicas of a deployment after it's created. Use this to scale resources up for performance or down for cost. To do this, call the `update` function on the `MLOpsScoringDeployment` instance with updated `kubernetes_options`.
- For more information on resource requests and limits in Kubernetes, see Resource Management for Pods and Containers.
- For more information about Pod Disruption Budget (PDB), see Pod Disruption Budget (PDB).

Enable or turn off model monitoring

To enable or turn off the model monitoring option, click the **Enable Monitoring** toggle. By default, the model monitoring option is turned off. If model monitoring is enabled, scoring data is saved.

Endpoint security

Select one of the following levels of endpoint security.

Note: The default security option can be configured at `securityOptions.default` in the `values.yaml` file (`charts/mlops/values.yaml`). By default, this is set to **Passphrase (stored as plain text)**.

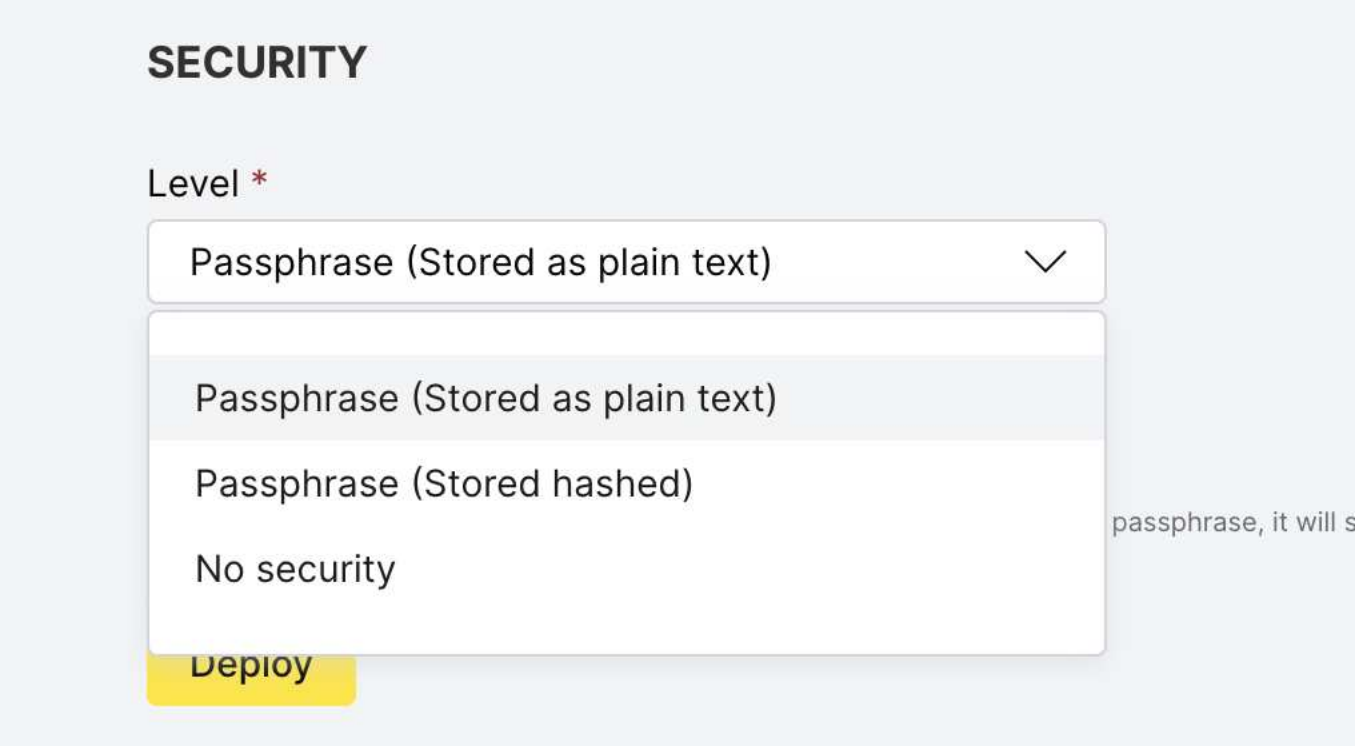
- **No security** - To skip configuration of endpoint security, select this option.
- **Passphrase (Stored as plain text)** - The passphrase is stored in plain text in the database. If this option is selected, the passphrase is visible in the UI after the model is deployed.
- **Passphrase (Stored hashed)** - The passphrase is stored as a hashed value in the database. If this option is selected, the passphrase is *not* visible in the UI after the model is deployed.
  - To support this option during deployment creation and update, add `PASSPHRASE_HASH_TYPE_PBKDF2` under `securityOptions.activated` in the `values.yaml` file (`charts/mlops/values.yaml`) . Note that having `PASSPHRASE_HASH_TYPE_BCRYPT` is neither sufficient nor required.

    ```
    securityOptions:
        activated:

            - .......
            - "PASSPHRASE_HASH_TYPE_PBKDF2"
            - .......
    ```

- **Token Authentication** - Supports multiple different OIDC token issuers. Note that this option requires the following additional configuration when deploying the H2O MLOps:
  - To support this option during deployment creation and update, update `securityOptions.activated` in the `values.yaml` file (`charts/mlops/values.yaml`) to include the value `AUTHORIZATION_PROTOCOL_OIDC` in the list:

    ```
    securityOptions:
        activated:

            - .......
            - "AUTHORIZATION_PROTOCOL_OIDC"
            - .......
    ```

Note: This option requires additional configuration, including setting `deployer.config.securityProxy.oidcIssuers` in the `values.yaml` file. This includes values such as the OIDC issuer URL and TLS credentials.



Driverless AI Deployment Wizard

The following steps describe how you can create H2O MLOps deployments directly from the completed experiment page in H2O Driverless AI.

Note:

To use this deployment option, you must specify `h2o_mlops_ui_url` in the `config.toml` file.

1. On the completed experiment page, click **Deploy**.
2. Click the **H2O MLOps** button. This action results in one of the following outcomes:
   - **The experiment is assigned to a single Project:** You are redirected to the Project detail page in the H2O MLOps app.
   - **The experiment is assigned to multiple Projects:** Select a project to go to in the H2O MLOps app. Alternatively, create a new Project to assign the experiment to. If you choose to create a new Project, you are prompted to enter a name and description for the Project. Once the new Project has been created and the experiment has been linked to it, you can click the **Go to MLOps page** button to navigate to the Project detail page in the H2O MLOps app.
   - **The experiment isn't assigned to any Project:** Select a Project to link the experiment to. Alternatively, create a new Project and link the experiment to it.

Deploying NLP models

H2O MLOps supports the deployment of Driverless AI and MLflow natural language processing (NLP) models.

Driverless AI

To deploy a Driverless AI NLP model, refer to the following steps:

1. In Driverless AI, pick a text dataset such as the Amazon Fine Food Reviews dataset or Airline Twitter Sentiment dataset.
2. Train a DAI model following the NLP-specific steps described in the documentation on NLP experiments in DAI.
3. Link the NLP experiment to an MLOps project. For more information, see the Driverless AI documentation on linking experiments.

4. Deploy as either a MOJO or a Python scoring pipeline. Note that using PyTorch or TensorFlow may make the MOJO unsupported.
5. The scorer is available at the endpoint URL provided by MLOps. You can use `curl` to test the endpoint. To see how this might look, refer to the following example request:

```
ububtu@ubuntu:/home/ubuntu$ curl -X POST -H "Content-Type: application/json" -
d @- DEPLOYMENT_ENDPOINT_URL << EOF
> {
>   "fields": [
>     "Description",
>     "Summary"
>   ],
>   "rows": [
>     [
>       "text",
>       "text"
>     ]
>   ]
> }
> EOF
{"fields":["PositiveReview.0","PositiveReview.1"],"id":"1c2ec2f0-74c7-11ec-ad8e-
3ee53b9e28aa","score":[["0.24813014","0.75186986"]]}nick@NPng-P5550:~/h2oworkspace/mlops-byom-
images/python-scorer$
```

MLflow

To deploy an MLflow NLP model, refer to the following steps:

1. Train your model.
2. Wrap the model in an MLflow model. For more information, see the example on uploading an MLflow custom Python model.
3. Upload and deploy the model using MLOps. For more information, see Deploying a model.

MLOps Processing Deployments

The following example demonstrates how to make an H2O MLOps deployment that implements custom logic to pre/post/chain process any number of other MLOps deployments.

```
import mlflow
import pandas
import requests
import typing
```

Processing Model Creation

Helper methods.

```
def convert_scores_to_pandas(
    scores_json: typing.Dict[str, typing.Any],
) -> pandas.DataFrame:
    return pandas.DataFrame(
        data=scores_json["score"], columns=scores_json["fields"]
    ).astype(float, errors="ignore")

def score_to_json(
    pdf: pandas.DataFrame,
    score_url: str,
    score_passphrase: typing.Optional[str],
    include_fields_in_output: typing.Optional[typing.List[str]]
) -> typing.Dict[str, typing.Any]:
```

```
    payload = dict(
        fields=list(pdf.columns),
        rows=pdf.fillna("").astype(str).to_dict("split")["data"],
    )
    if include_fields_in_output:
        payload["includeFieldsInOutput"] = include_fields_in_output
    result = requests.post(
        url=score_url, json=payload, headers={"Authorization": f"Bearer {score_passphrase}"}
    )
    result.raise_for_status()
    return result.json()

def score_to_pandas(
    pdf: pandas.DataFrame,
    score_url: str,
    score_passphrase: typing.Optional[str],
    include_fields_in_output: typing.Optional[typing.List[str]]
) -> typing.Dict[str, typing.Any]:
    scores_json = score_to_json(
        pdf, score_url, score_passphrase, include_fields_in_output
    )
    return convert_scores_to_pandas(scores_json)
```

MLflow model definition example. Note that you can implement any pre, post, chaining, etc. logic, utilizing the `score` helper methods to get results from other models.

```
class CCProcessor(mlflow.pyfunc.PythonModel):
    """Converts output to binary result based on threshold."""

    def predict(self, context, model_input) -> pandas.DataFrame:
        target_column = "default payment next month"
        id_column = "ID"
        threshold = 0.3

        scores_dataframe = score_to_pandas(
            model_input,
            score_url="https://model.internal.dedicated.h2o.ai/3ae169a7-21a6-419b-bbf2-
bce0ef3463bc/model/score",
            score_passphrase="j3UyU2PLB6k3tnpYW1CfusDXPgm72mu2kaR2elxL9jA",
            include_fields_in_output=[id_column]
        )

        processed_dataframe = pandas.DataFrame()
        processed_dataframe[id_column] = scores_dataframe[id_column].astype(int)
        processed_dataframe[
            f"{target_column} | threshold={threshold}"
        ] = scores_dataframe[f"{target_column}.1"] > threshold
        return processed_dataframe
```

Test predict method.

```
x = pandas.read_csv("/Users/jgranados/datasets/creditcard.csv")[:10]
y = CCProcessor().predict(context=None, model_input=x)
y
```

	ID	default payment next month | threshold=0.3
0	1	True
1	2	True
2	3	False
3	4	False

	ID	default payment next month | threshold=0.3
4	5	False
5	6	False
6	7	False
7	8	False
8	9	True
9	10	False

Define schema (required) using dataframes from testing.

```
model_signature = mlflow.models.signature.infer_signature(x,y)
model_signature
```

```
    inputs:
      ['ID': long, 'LIMIT_BAL': long, 'SEX': long, 'EDUCATION': long, 'MARRIAGE': long, 'AGE': long, 'PAY_0':
    outputs:
      ['ID': long, 'default payment next month | threshold=0.3': boolean]
```

Save processor model.

```
model_path = "processor"

mlflow.pyfunc.save_model(
    path=model_path,
    python_model=CCProcessor(),
    signature=model_signature,
)
```

Test MLFlow can load and score the saved model.

```
model = mlflow.pyfunc.load_model(model_path)
print(model.metadata.get_input_schema())
print(model.metadata.get_output_schema())
model.predict(x)
```

```
    ['ID': long, 'LIMIT_BAL': long, 'SEX': long, 'EDUCATION': long, 'MARRIAGE': long, 'AGE': long, 'PAY_0': l
    ['ID': long, 'default payment next month | threshold=0.3': boolean]
```

	ID	default payment next month | threshold=0.3
0	1	True
1	2	True
2	3	False
3	4	False
4	5	False
5	6	False
6	7	False
7	8	False
8	9	True
9	10	False

Make archive to upload and deploy using MLOps UI or Python client.

```
import shutil

zip_path = shutil.make_archive(
    "processor_mlflow", "zip", "processor"
)
```

Scoring

Assumes archive has been deployed on MLOps.

Deployment to be processed Example of unprocessed output.

```
result = requests.post(
    url="https://model.internal.dedicated.h2o.ai/3ae169a7-21a6-419b-bbf2-bce0ef3463bc/model/score",
    json=dict(
        fields=list(x.columns),
        rows=x.fillna("").astype(str).to_dict("split")["data"],
    ),
    headers={"Authorization": f"Bearer j3UyU2PLB6k3tnpYW1CfusDXPgm72mu2kaR2elxL9jA"}
)
result.raise_for_status()
result.json()
```

```
    {'fields': ['default payment next month.0', 'default payment next month.1'],
     'id': '29644d94-69ce-11ed-8dd0-42a220533d6d',
     'score': [['0.5026670558588076', '0.4973329441411924'],
      ['0.5757092185564415', '0.4242907814435585'],
      ['0.7779584616197291', '0.22204153838027083'],
      ['0.7571721069431696', '0.24282789305683036'],
      ['0.8100351192319214', '0.1899648807680786'],
      ['0.7414997325267836', '0.25850026747321647'],
      ['0.8639312267025963', '0.13606877329740377'],
      ['0.7468108781657142', '0.25318912183428577'],
      ['0.6648249807372442', '0.3351750192627559'],
      ['0.7785471596551806', '0.22145284034481938']]}
```

Processing deployment Example of processed output.

```
requests.get(
    "https://model.internal.dedicated.h2o.ai/c51d99b8-1563-440f-9868-7b23353bb402/model/schema"
).json()["schema"]["outputFields"]
```

```
    [{'name': 'ID', 'dataType': 'Int64'},
     {'name': 'default payment next month | threshold=0.3', 'dataType': 'Bool'}]
```

```
result = requests.post(
    url="https://model.internal.dedicated.h2o.ai/c51d99b8-1563-440f-9868-7b23353bb402/model/score",
    json=dict(
        fields=list(x.columns),
        rows=x.fillna("").astype(str).to_dict("split")["data"],
    )
)
result.raise_for_status()
result.json()
```

```
    {'fields': ['ID', 'default payment next month | threshold=0.3'],
     'id': '3ee67d43-5219-48dc-a02f-cea2b4e5c49d',
     'score': [['1', 'True'],
      ['2', 'True'],
      ['3', 'False'],
      ['4', 'False'],
      ['5', 'False'],
      ['6', 'False'],
      ['7', 'False'],
      ['8', 'False'],
      ['9', 'True'],
      ['10', 'False']]}
```

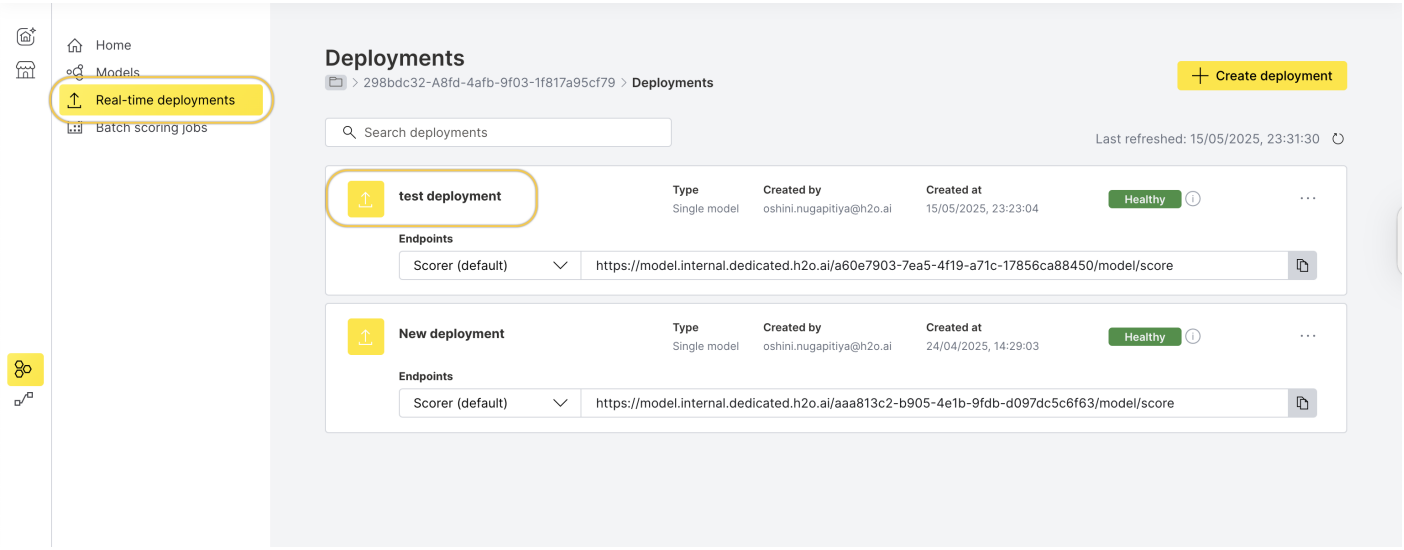

View deployments

This page explains how to view and manage deployments in H2O MLOps. It covers how to access deployment details, monitor performance, perform scoring, and carry out common tasks such as downloading logs or deleting deployments.

To view the list of available deployments, click **Real-time deployments** on the left navigation menu.

Understand the Deployments page

To view details of a specific deployment, click the name of the deployment you want to view.



The **Deployment details** page is divided into the following three tabs:

- Details
- Endpoints
- Quick scoring

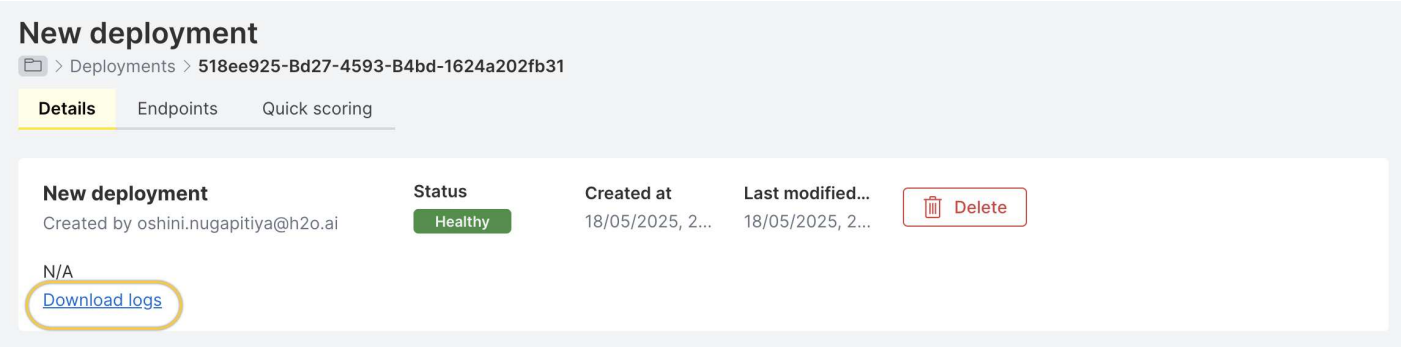
Details

The **Details** tab provides key information about the deployment, including configuration, status, and advanced settings.

- **Deployment name:** The name of to the deployment.
- **Deployment description:** A brief summary of the deployment's purpose or functionality.
- **Deployment type:** The deployment type for the deployment (Single model, A/B test, or champion/challenger).
- **Status:** The current status of the deployment. For more information, see States.
- **Deployed model details:** Information about the model associated with the deployment:
  - Model name: The name of the deployed model.
  - Version: The specific version of the model in use.
  - Artifact type and runtime: The artifact type and runtime for the deployment.
- **Advanced settings:** Additional configuration options for the deployment. You can update the following:
  - Kubernetes options
    - Replicas: The number of static replicas of the deployed model.
    - Requests: The amount of resources the deployment requires from Kubernetes.
    - Limits: The maximum amount of resources that are available to the deployment.
  - Model monitoring: To enable or disable model monitoring, use the **Enable Monitoring** toggle. When enabled, scoring data is collected and stored.
  - Security
    - Level: The security level specified when the deployment was created.
    - Passphrase: If you selected the **Passphrase (Stored as plain text)** option when creating the deployment, the passphrase can be viewed in the Security Details section. Note that if you select the **Passphrase (Stored hashed)** option, the passphrase cannot be viewed.

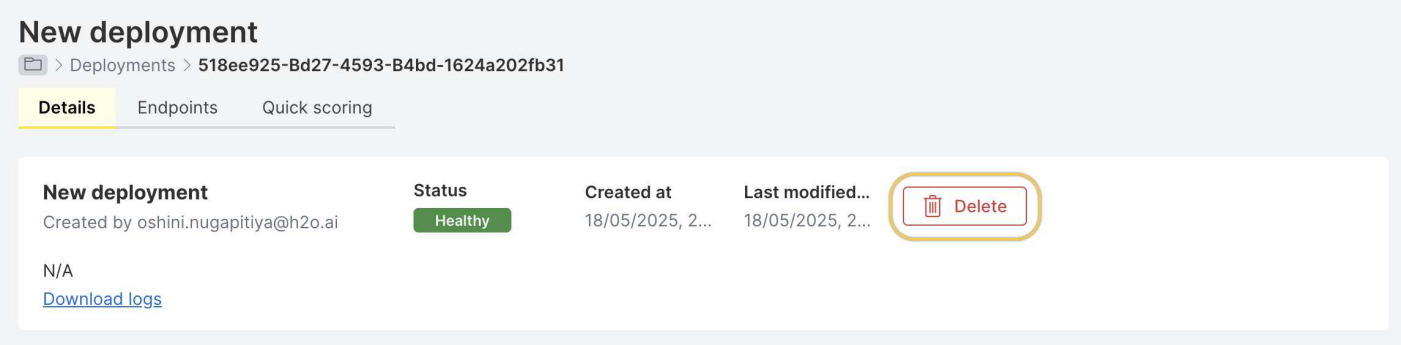
  After updating the advanced settings, click **Save changes**.
  For more information on the advanced settings, see Advanced settings.

Download deployment logs To download deployment logs, click **Download Logs** in the **Details** tab.



The logs are downloaded to your device as a `ZIP` file.

Delete a deployment To delete a deployment, click the **Delete** button in the **Details** tab.



States The following is a list of possible states for deployments in MLOps:

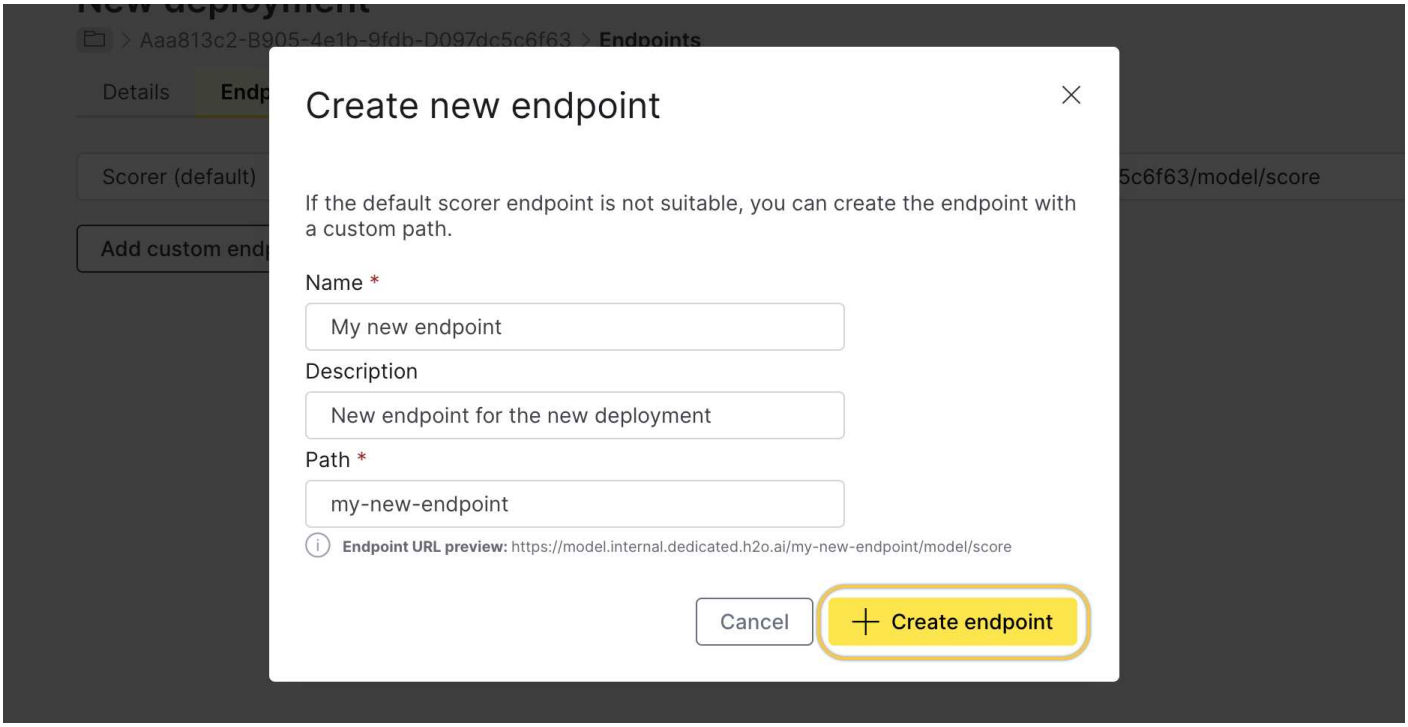
- **Preparing** - The deployment is being prepared for launch
- **Launching** - The deployment is launching to an environment
- **Failed** - The deployment failed during preparation or launch
- **Healthy** - The deployment is alive and healthy
- **Unhealthy** - Health issues have been detected with the launched deployment
- **Terminating** - The deployment is terminating
- **Pending** - The deployment has been created and is awaiting processing
- **Stopped** - The deployment is scaled down.

Endpoints

From the **Endpoints** tab, you can view and copy the default deployment URL. If the default scorer endpoint is not suitable, you can add a custom endpoint with a different path.

Add custom endpoint

1. Click **Add custom endpoint**.
2. Enter a name for the new endpoint.
3. (Optional) Add a description for the endpoint.
4. Specify the endpoint path.
5. Click **Create endpoint**.

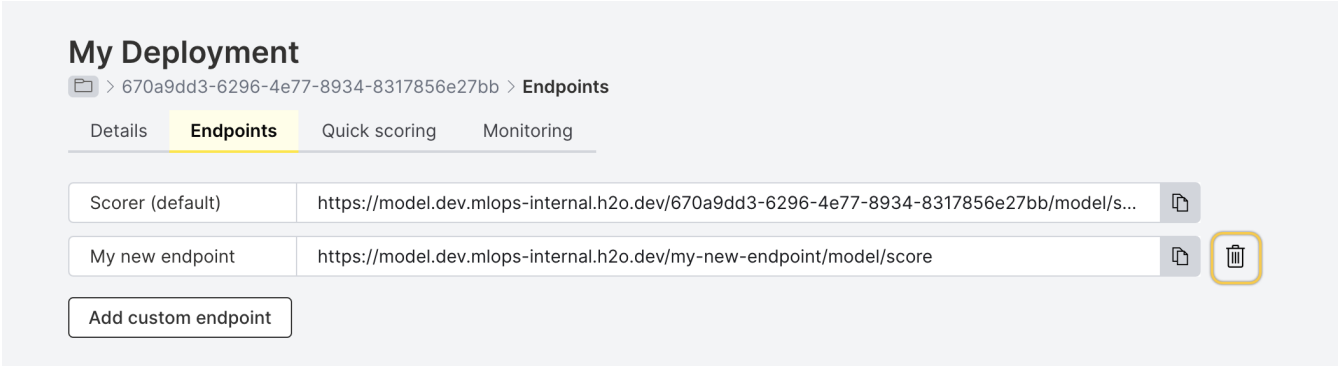


Scoring with endpoint URL In the **Endpoints** tab, copy the endpoint URL.

You can use the URL in your own application for sending scoring requests.

Delete an endpoint warning Deleting an endpoint is permanent and can't be undone.

1. Click delete next to the endpoint.



2. Click **Delete** to confirm.

Quick scoring

To learn how to perform quick scoring, see Quick scoring.

Missing values

To indicate a field as a missing value for any of the following runtime options, use the corresponding value displayed in the following table.

Note: The information provided in this section is only guaranteed to work if no changes have been made to the default `missing_values` configuration setting in Driverless AI (DAI). If you have changed the `missing_values` configuration setting in DAI, contact the H2O support team for assistance.

Runtime	Numeric missing value	String missing value
Driverless AI MOJO scorer	Empty string	Empty string
Python Pipeline scorer (MLOps 0.57.3 and earlier)	`"1.7976931348623157e+308"`	Empty string
Python Pipeline scorer (MLOps 0.58.0 and later)	`"1.7976931348623157e+308"` or empty string	Empty string
H2O-3 MOJO	Empty string	Empty string
MLflow/ `.pkl` file	Not supported	Empty string

Vertical Pod Autoscaler (VPA) support

Vertical Pod Autoscaler (VPA) is supported in the Deployer. VPA allows dynamic scaling of CPU and memory resources based on application usage, improving resource efficiency and optimizing costs.

For more information, see the official VPA GitHub README.

Note: For a list of known limitations, see the Known limitations section of the VPA GitHub README.

Configurations

The following configurations provide control over resource usage based on VPA settings.

```
vpa:
  # Whether to enable the VPA.
  enabled: false
  # The CPU max threshold for the VPA.
  cpuMaxThreshold: 0
  # The CPU unit for the VPA.
  # Available units for CPU:
  # - CORES: For CPU cores
  # - MILLICORES: For CPU millicores
  cpuMaxThresholdUnit: "MILLICORES"
  # The memory max threshold for the VPA.
  memoryMaxThreshold: 0
  # The memory unit for the VPA.
  # Available units for memory:
  # - MIB: For memory in MiB
  # - GIB: For memory in GiB
  memoryMaxThresholdUnit: "MIB"
```

Pod Disruption Budget (PDB)

This page provides a detailed overview of the Pod Disruption Budget (PDB) API specification and its integration with H2O MLOps. A Pod Disruption Budget is a Kubernetes resource that specifies the minimum number of pods that must remain available during a disruption caused by voluntary actions (like scaling down) or involuntary actions (like node failures). PDBs help maintain application stability by preventing too many pods from being simultaneously unavailable. For more information about PDBs, see Disruptions.

PDB API specification

```
// Represents configuration for Pod Disruption Budget (PDB).
// Only one of the two options; min_available or max_unavailable should be specified.
message PodDisruptionBudgetSpec {
  // Only one of these disruption policies can be specified
  oneof disruption_policy {
    // The minimum number of pods that must be available after the eviction
    MinAvailable min_available = 1;
    // The maximum number of pods that can be unavailable after the eviction
    MaxUnavailable max_unavailable = 2;
  }
}
```

The `PodDisruptionBudgetSpec` defines the configuration for PDB. It includes a `disruption_policy`, where users can specify either `min_available` or `max_unavailable`. Only one policy can be specified at a time. The `min_available` policy represents the minimum number of pods that must be available after the eviction. The `max_unavailable` policy represents the maximum number of pods that can be unavailable after the eviction.

```
// Represents minimum availability configuration
message MinAvailable {
  oneof value {
    int32 pods = 1;        // Absolute number of pods
    int32 percentage = 2;  // Percentage of pods
  }
}
```

`MinAvailable` specifies the minimum availability of configuration.

```
// Represents maximum unavailability configuration
message MaxUnavailable {
  oneof value {
    int32 pods = 1;        // Absolute number of pods
    int32 percentage = 2;  // Percentage of pods
  }
}
```

`MaxUnavailable` specifies the maximum unavailability of configuration.

Helm chart configuration

To enable PDB globally or for specific deployments, configure the Helm chart using the following parameters:

```
podDisruptionBudget:
    # -- Whether to enable the PodDisruptionBudget globally.
    # -- PS: This does not deploy a PDB for each deployment by default,
    # -- instead it will just give user the ability to set a PDB for each deployment.
    # -- If enabled, it will be possible to set a PDB for each deployment.
    # -- If not, it won't be possible to set a PDB per deployment.
    # -- PS: PDB should not be enabled for a deployment if VPA is enabled for that specific deployment.
    enabled: false
```

The `enabled` flag determines whether PDB can be configured for deployments. If enabled, users can set specific PDB configurations for each deployment. PDB should be disabled if Vertical Pod Autoscaler (VPA) is enabled for the deployment.

Understand model scoring

Model scoring is the process of using a deployed model to generate predictions based on input data. In H2O MLOps, once a model version is deployed, you can send data to the deployment and receive predictions in response.

H2O MLOps provides multiple ways to perform scoring:

- Quick scoring: A UI-based option to test your deployment with sample input data.
- Deployment scorer: A Python client–based option for scoring against deployments.

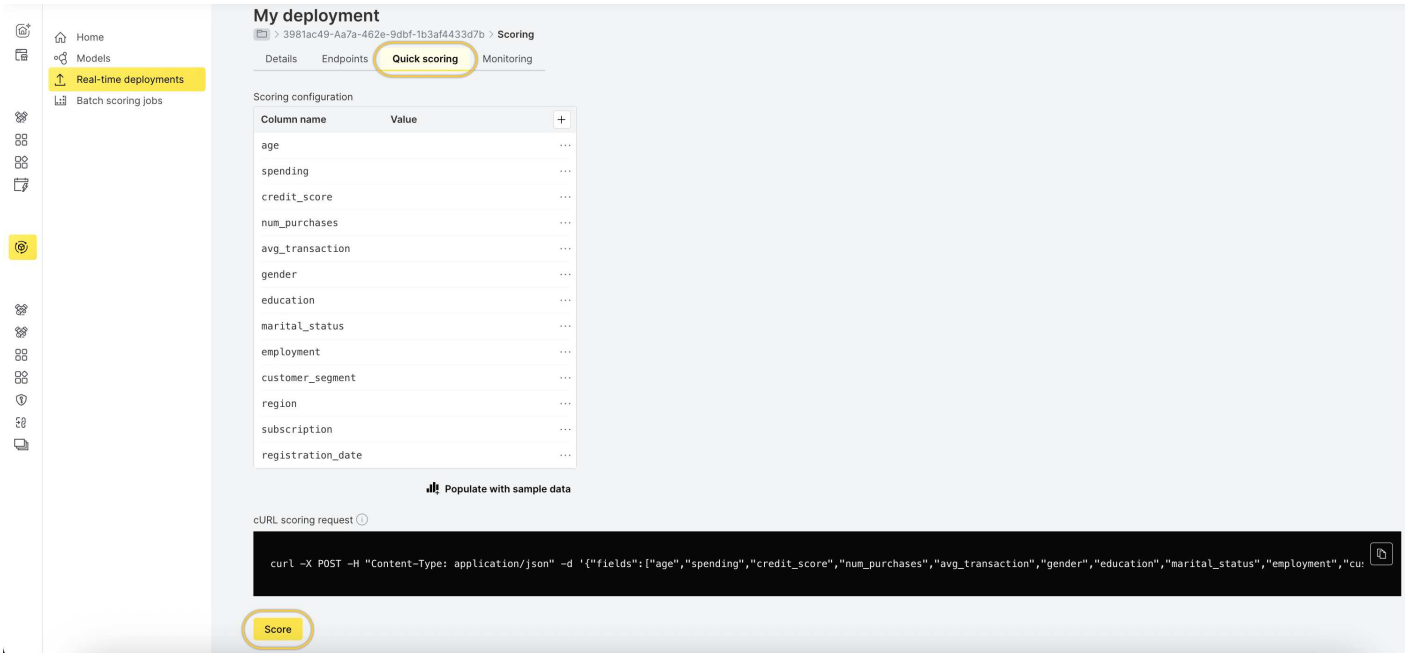
To learn more about scoring methods and how to use them, see:

- Quick scoring
- Deployment scorer

Quick scoring

Score directly from the UI

To score the deployment directly from the UI, click **Score** in the **Quick scoring** tab.



After scoring, the results are displayed in the interface.

Scoring with cURL scoring request

The cURL scoring request is auto-generated based on the scoring configuration.

1. In the **Quick scoring** tab, copy the sample `curl` scoring request.

2. Open a Terminal window and paste the sample `curl` request to view the model scoring information.

Shapley values support

Shapley value support in MLOps requires a model created with H2O-3 or Driverless AI version 1.10 or later, and is available for both MOJO and Python pipeline artifacts.

The following steps describe how to enable and request Shapley values.

Step 1: Enable Shapley values when deploying a model

For DAI experiments

If the **Driverless AI MOJO pipeline** artifact type (`dai/mojo_pipeline`) is selected when deploying a model, several runtimes that enable support for Shapley values are available. The selected runtime determines the type of Shapley value you can request in the following step. Depending on the selected runtime option, deploying with Shapley support doubles or triples the RAM requirements of the runtime.

Note: You can skip this step if you're using a DAI Python pipeline.

- **H2O.ai MOJO scorer with Shapley values for transformed features** (`mojo_runtime_shapley_transformed`): Generate Shapley values for features or columns that have been transformed by DAI.
- **H2O.ai MOJO scorer with Shapley values for all features** (`mojo_runtime_shapley_all`): Generate Shapley values for *either* original or transformed features.
- **H2O.ai MOJO scorer with Shapley values for original features** (`mojo_runtime_shapley_original`): Generate Shapley values for features or columns that existed as part of the original dataset or experiment.

For H2O-3 MOJO experiments

If the H2O-3 MOJO artifact type (`h2o3/mojo`) is selected when deploying a model, only one runtime is available that enables support for Shapley values: `h2o3_mojo_runtime_shapley_transformed`.

- **H2O-3 MOJO scorer with Shapley values for transformed features** (`h2o3_mojo_runtime_shapley_transformed`): Generate Shapley values for features or columns transformed during training.

Note:

- H2O MLOps supports Shapley `TRANSFORMED` values in H2O-3 for MOJO experiments.
- Shapley values are supported only for the following model types:
  - DRF
  - GBM
  - XGBoost
- Shapley values are **not** supported for the following models:
  - Multinominal classification models
  - Binominal models with Binominal Double Trees parameter set
  - GLM models

Step 2: Request Shapley values in a `curl` request

By default, Shapley values aren't returned in a `curl` request. To get Shapley values (that is, the Shapley type enabled in the preceding step), you must include the `requestShapleyValueType` argument in the `curl` request and set the value as either `ORIGINAL` or `TRANSFORMED`.

Note:

- The specified value must correlate with the runtime selected in the preceding step.
- The following steps describe how to check which Shapley values have been enabled:
  1. Copy the endpoint URL of the deployment.
  2. In the endpoint URL, replace `/score` with `/capabilities`.
  3. Paste the endpoint URL in a browser window. One to three different terms are displayed that indicate whether the deployment supports Shapley values for original features and/or transformed features: [SCORE, CONTRIBUTION_ORIGINAL, CONTRIBUTION_TRANSFORMED]

- To try this using the H2O MLOps Python client, see View scorer capabilities.

`ORIGINAL`

`ORIGINAL` enables the ability to generate Shapley values for features or columns that existed as part of the original dataset or experiment.

Example usage:

```
"requestShapleyValueType": "ORIGINAL"
```

`TRANSFORMED`

`TRANSFORMED` enables the ability to generate Shapley values for features or columns that have been transformed by DAI.

Example usage:

```
"requestShapleyValueType": "TRANSFORMED"
```

Note:

- By default, this value is set to `NONE`, which is the equivalent of not providing the `requestShapleyValueType` argument in the `curl` request.
- To try this using the H2O MLOps Python client, see Shapley values.

The following is a sample `curl` request and response with Shapley values enabled for original features:

```
curl -X POST -H "Content-Type: application/json" -d @- <SCORING_ENDPOINT_URL> << EOF
    {
    "fields": [
        "LIMIT_BAL",
        "SEX",
        "EDUCATION",
        "MARRIAGE",
        "AGE",
        "PAY_0",
        "PAY_2",
        "PAY_3",
        "PAY_4",
        "PAY_5",
        "PAY_6",
        "BILL_AMT1",
        "BILL_AMT2",
        "BILL_AMT3",
        "BILL_AMT4",
        "BILL_AMT5",
        "BILL_AMT6",
        "PAY_AMT2",
        "PAY_AMT3",
        "PAY_AMT4",
        "PAY_AMT5",
        "PAY_AMT6"
    ],
    "rows": [
        [
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
```

```
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0",
        "0"
        ]
    ], "requestShapleyValueType": "ORIGINAL"
    }
    EOF

    {
        "featureShapleyContributions":
        {
            "contributionGroups":
            [
                {
                    "contributions":
                    [
                        [
                            "0.3031580540597976",
                            "0.05037104158009451",
                            "0.01197491002508829",
                            "-0.09613404645427222",
                            "0.03349942127192829",
                            "-0.19629869420775475",
                            "-0.05457586577961132",
                            "-0.016488709633310006",
                            "0.03022179422131117",
                            "-0.010334700480366232",
                            "-0.001831092307318766",
                            "0.24078515169214335",
                            "0.059130207887999234",
                            "-0.03856367964221913",
                            "-4.1371086615778267E-4",
                            "0.02549452684520287",
                            "-0.012600605263304759",
                            "0.17373018794999764",
                            "0.19368473683713824",
                            "0.1887846519524733",
                            "0.08455862402217218",
                            "0.04817053716929957",
                            "-1.4823175495435195"
                        ]
                    ]
                }
            ],
            "features":
            [
                "contrib_LIMIT_BAL",
                "contrib_SEX",
```

```
                "contrib_EDUCATION",
                "contrib_MARRIAGE",
                "contrib_AGE",
                "contrib_PAY_0",
                "contrib_PAY_2",
                "contrib_PAY_3",
                "contrib_PAY_4",
                "contrib_PAY_5",
                "contrib_PAY_6",
                "contrib_BILL_AMT1",
                "contrib_BILL_AMT2",
                "contrib_BILL_AMT3",
                "contrib_BILL_AMT4",
                "contrib_BILL_AMT5",
                "contrib_BILL_AMT6",
                "contrib_PAY_AMT2",
                "contrib_PAY_AMT3",
                "contrib_PAY_AMT4",
                "contrib_PAY_AMT5",
                "contrib_PAY_AMT6",
                "contrib_bias"
            ]
        },
        "fields":
        [
            "default payment next month.0",
            "default payment next month.1"
        ],
        "id": "f0395bc4-47d0-11ec-b7eb-fad6d6e23f65",
        "score":
        [
            [
                "0.6144353",
                "0.38556466"
            ]
        ]
    }
```


Test Time Augmentation (TTA) support

Driverless AI supports rolling-window-based predictions for time series experiments using Test Time Augmentation (TTA). TTA is only available for Python Scoring Pipeline artifacts. This page describes support for TTA in H2O MLOps.

Step 1: Enable TTA when deploying a model

If the **Driverless AI Python scoring pipeline** artifact type is selected when deploying a model, Test Time Augmentation will automatically be enabled for capable models.

Step 2: Check if the deployment has TTA support in a `curl` request

The following is a sample `curl` request and response to check depoyment capabilities of a Test Time Augmentation enabled deployment:

```
curl -X GET https://<SCORER_API_BASE_URL>/model/capabilities
```

```
["SCORE","CONTRIBUTION_ORIGINAL","CONTRIBUTION_TRANSFORMED","TEST_TIME_AUGMENTATION"]
```

Note: `TEST_TIME_AUGMENTATION` must be present in the cpabilities response for Test Time Augmentation scoring to work.

To try this using H2O MLOps Python client, see View scorer capabilities.

Step 3: Score in a `curl` request

TTA requires passing known historical target values in the request.

The following is a sample `curl` scoring request and response that will trigger TTA (in this example we are forecasting weekly sales):

```
curl -X POST -H "Content-Type: application/json" -d @- <SCORING_ENDPOINT_URL> << EOF
{
"fields": [
    "Store",
    "Dept",
    "Date",
    "IsHoliday",
    "Weekly_Sales"
],
"rows": [
    ["1", "1", "2011-06-24", "0", "15682.81"],
    ["1", "1", "2011-07-01", "0", "15363.5"],
    ["1", "1", "2011-07-08", "0", "16148.87"],
    ["1", "1", "2011-07-15", "0", "15654.85"],
    ["1", "1", "2011-07-22", "0", "15766.6"],
    ... <intermediate rows omitted> ...
    ["1", "1", "2012-09-28", "0", "18947.81"],
    ["1", "1", "2012-10-05", "0", "21904.47"],
    ["1", "1", "2012-10-12", "0", "22764.01"],
    ["1", "1", "2012-10-19", "0", "24185.27"],
    ["1", "1", "2012-10-26", "0", "27390.81"],
    ["1", "1", "2012-11-02", "0", ""],
    ["1", "1", "2012-11-09", "0", ""]
], "includeFieldsInOutput": ["Store", "Dept", "Date"]
}
EOF

{
"fields":
[
    "Weekly_Sales",
    "Weekly_Sales.lower",
```

```
    "Weekly_Sales.upper",
    "Store",
    "Dept",
    "Date",
],
"id": "70e97f86-133e-11ed-9b13-9e9aacfaa41c",
"score": [
    ["15850.026", "-4002.4613906250015", "34435.428949218745", "1", "1", "2011-06-24"],
    ["16170.747", "-3681.7406875000015", "34756.149652343745", "1", "1", "2011-07-01"],
    ["15866.074", "-3986.4135390625015", "34451.476800781245", "1", "1", "2011-07-08"],
    ["16254.93", "-3597.5580703125015", "34840.332269531245", "1", "1", "2011-07-15"],
    ["15990.216", "-3862.2719375000015", "34575.618402343745", "1", "1", "2011-07-22"],
    ... <intermediate rows omitted> ...
    ["18941.385", "-911.1029921875015", "37526.787347656245", "1", "1", "2012-09-28"],
    ["18942.018", "-910.4701796875015", "37527.420160156245", "1", "1", "2012-10-05"],
    ["21030.857", "1178.3696640624985", "39616.260003906245", "1", "1", "2012-10-12"],
    ["21969.998", "2117.5102890624985", "40555.400628906245", "1", "1", "2012-10-19"],
    ["23745.594", "3893.1059921874985", "42330.996332031245", "1", "1", "2012-10-26"],
    ["26146.97", "6294.4829453124985", "44732.373285156245", "1", "1", "2012-11-02"],
    ["13077.371", "-6775.1166640625015", "31662.773675781245", "1", "1", "2012-11-09"]
]
}
```


Note: In the preceding example, the last two rows contain the predictions of interest. The rest of the rows also have predictions (rather than the input value) and can be compared against the known target values to evaluate model accuracy.

Prediction intervals support

To enable support for prediction intervals in H2O MLOps, set `requestPredictionIntervals` parameter to `true`. Note that if prediction intervals are not supported by the model or not returned for some reason, H2O MLOps will either leave the field empty or return an error response. This mechanism ensures that you are aware when prediction intervals are not available.

Once enabled, the prediction intervals are returned as an array. For each prediction, a lower and upper bound are returned.

Note: Prediction interval support is currently only available for regression models in all MOJO runtimes and the Driverless AI scoring pipeline runtime.

Step 1: Check if the deployment has `requestPredictionIntervals` support in a curl request

Use the `/capabilities` endpoint to confirm that the model supports prediction intervals.

```
curl -X GET https://<DEPLOYMENT_URL>/model/capabilities
```

```
["SCORE_PREDICTION_INTERVAL","SCORE","CONTRIBUTION_ORIGINAL","CONTRIBUTION_TRANSFORMED"]
```

The `SCORE_PREDICTION_INTERVAL` capability indicates that prediction intervals are supported.

To try this using H2O MLOps Python client, see View scorer capabilities.

Step 2: Make a prediction with `requestPredictionIntervals` enabled

To request prediction intervals using the H2O MLOps Python client, see Prediction intervals.

Sample output:

```
{'fields': ['score'],
 'id': '1b0488b6-ee91-11ed-a05d-4ab989c17db4',
 'predictionIntervals': {'fields': ['score.lower',
   'score.upper'],
  'rows': [['55524.044704861124', '371673.57907986105']]},
 'score': [['200005.90798611112']]}
```

H2O MLOps Scoring REST API: OpenAPI specification file

The **H2O MLOps OpenAPI specification file** outlines the structure and functionality of the H2O MLOps Scoring REST API, detailing available endpoints, request and response formats, and authentication requirements.

- **OpenAPI specification**: Access the specification in YAML format:
  - Download OpenAPI Spec (YAML)

Model monitoring

H2O MLOps model monitoring involves observing the performance and behavior of deployed models to ensure they continue to operate effectively and to identify issues such as model drift.

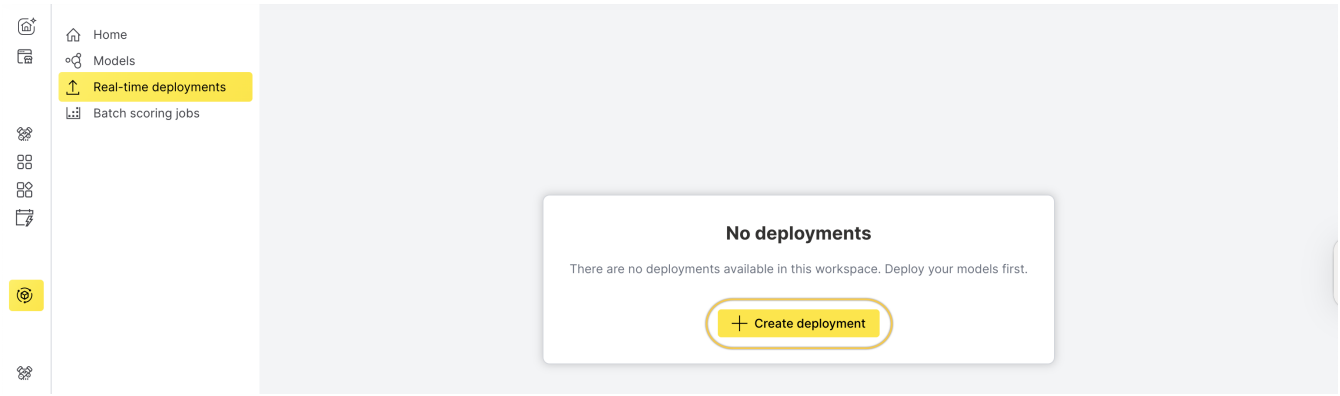
This guide explains how to configure model monitoring during deployment, analyze aggregated data, and identify model drift. Follow the steps below to set up and use model monitoring in H2O MLOps.

Model monitoring with the UI

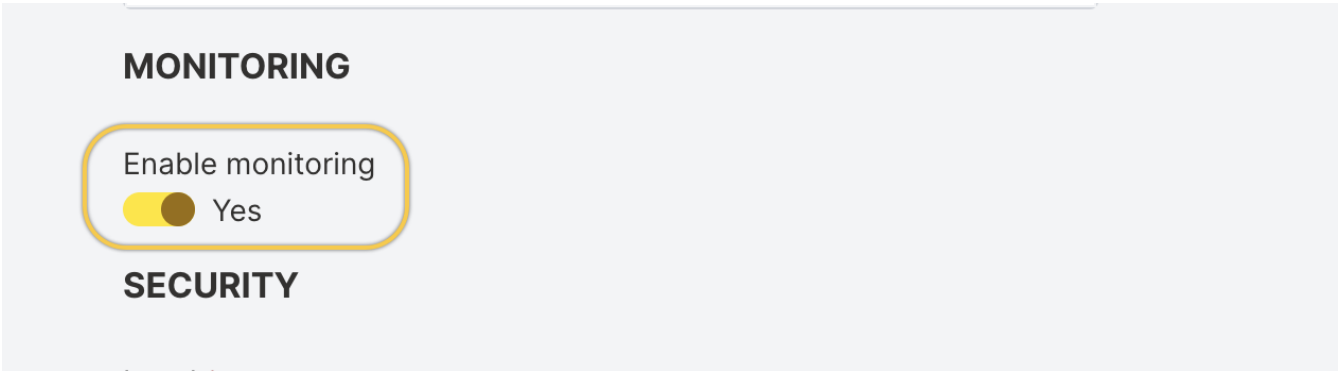
Step 1: Enable model monitoring

To enable model monitoring for your deployment:

1. In the left navigation panel, click **Real-time deployments**.
2. Click **Create deployment**.



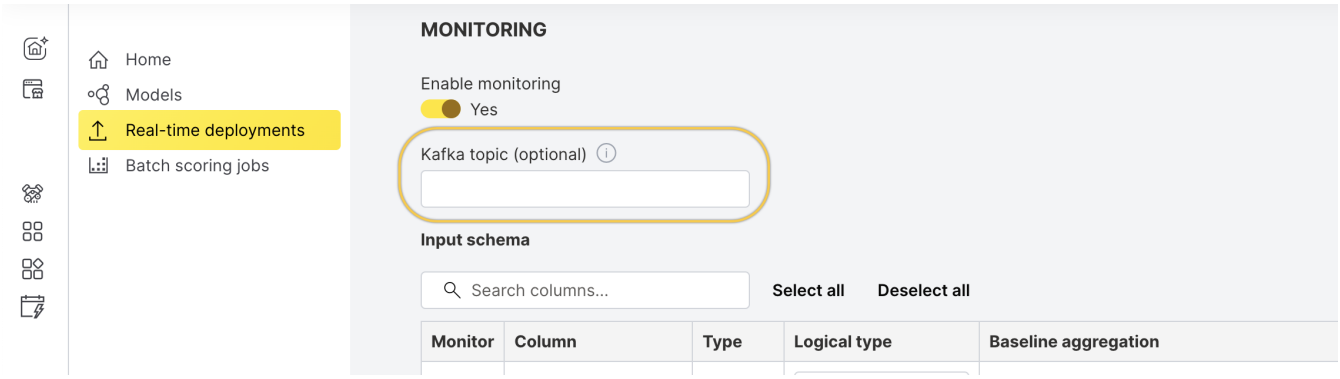
3. On the **Create new deployment** page, click **Advanced settings**.
4. Toggle **Enable monitoring** to **Yes**.



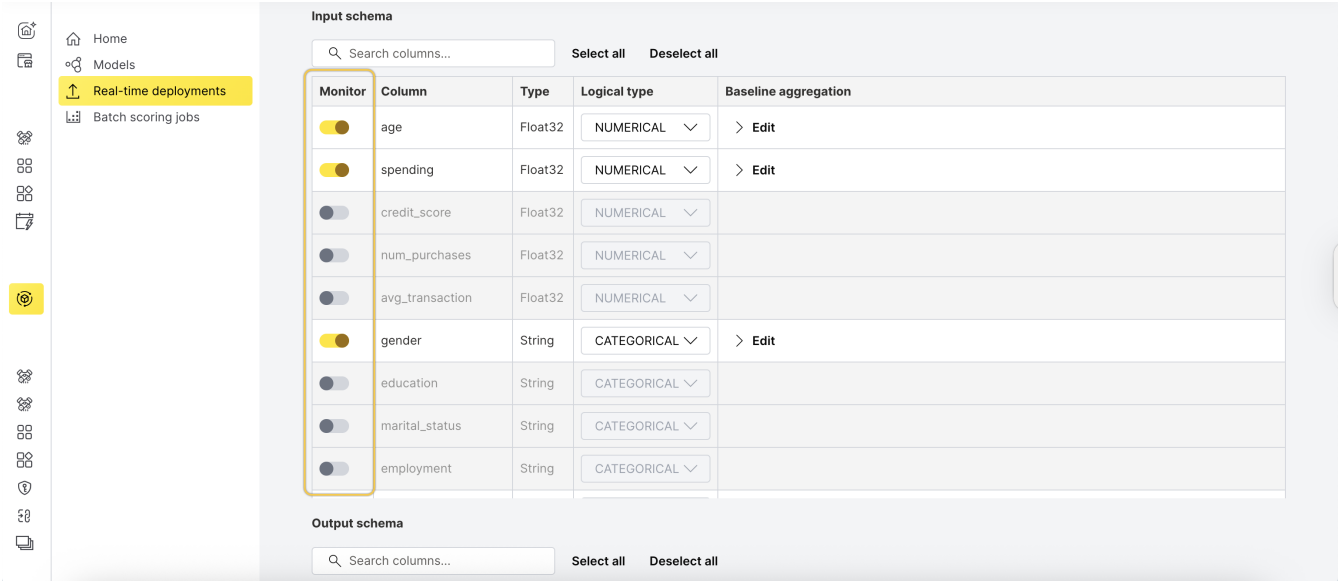
Step 2: Configure and deploy

During model deployment, configure monitoring to collect and analyze data.

1. If Kafka is available in your environment, provide a pre-created Kafka topic where raw data will be sent. For more information, see Raw data export to Kafka.

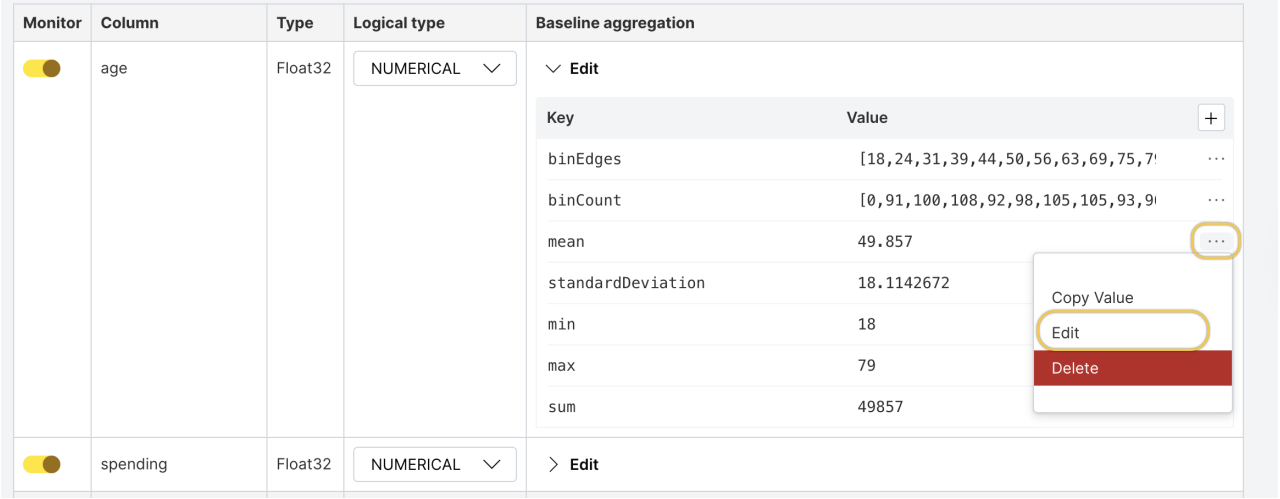


2. Select the columns you want to monitor.

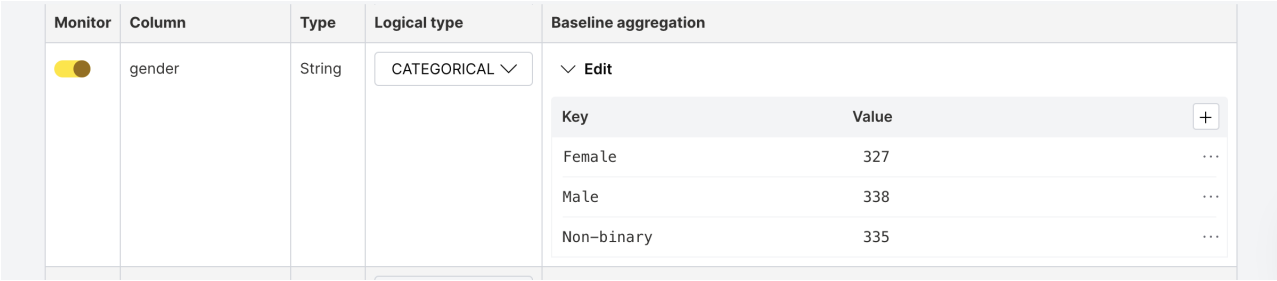


3. Provide baseline data for comparison:

- **Numerical features**:



- **Categorical features**:



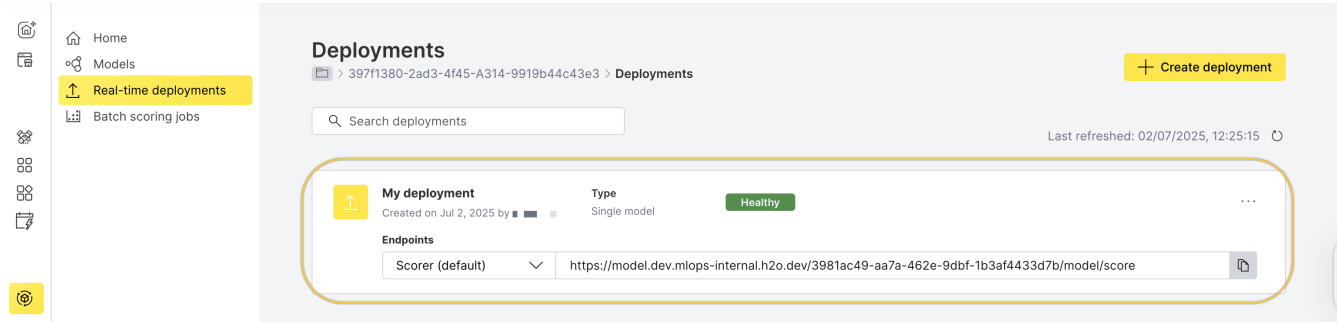
4. Click **Deploy**.

Step 3: Start scoring

Once the deployment is **Healthy**, you can begin scoring.

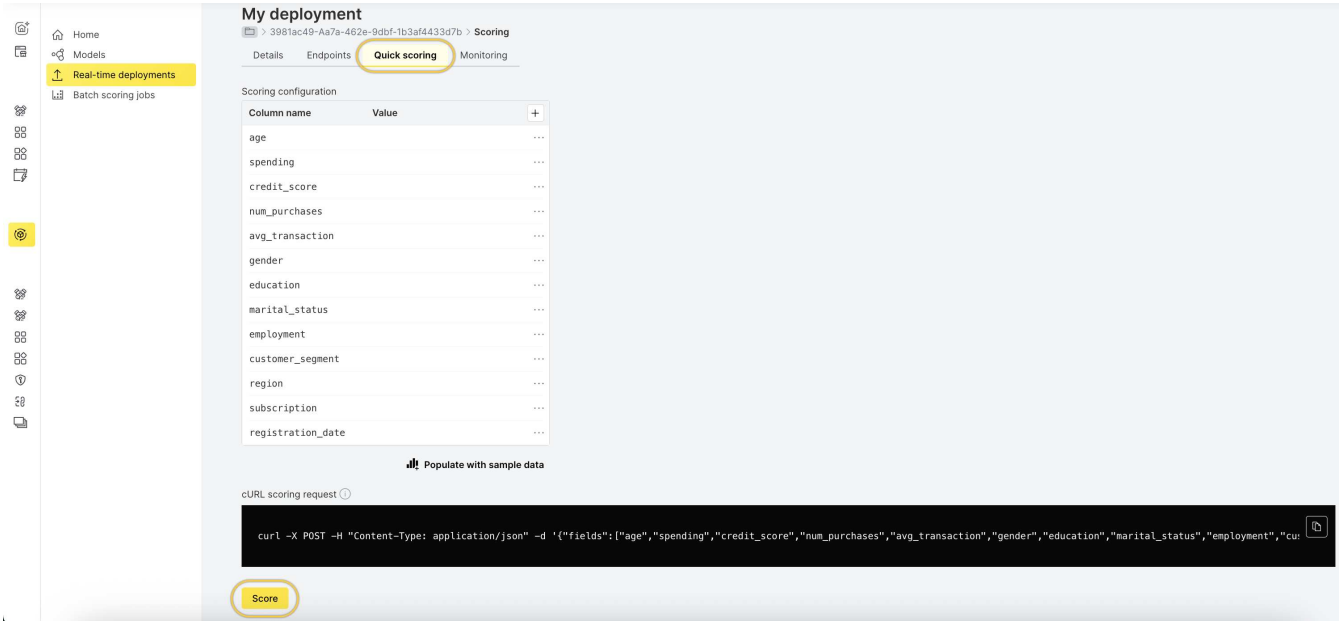
1. In the left navigation panel, click **Real-time deployments**.

2. Select the deployment you created.



3. Go to the **Quick scoring** tab.

4. Click **Score**.

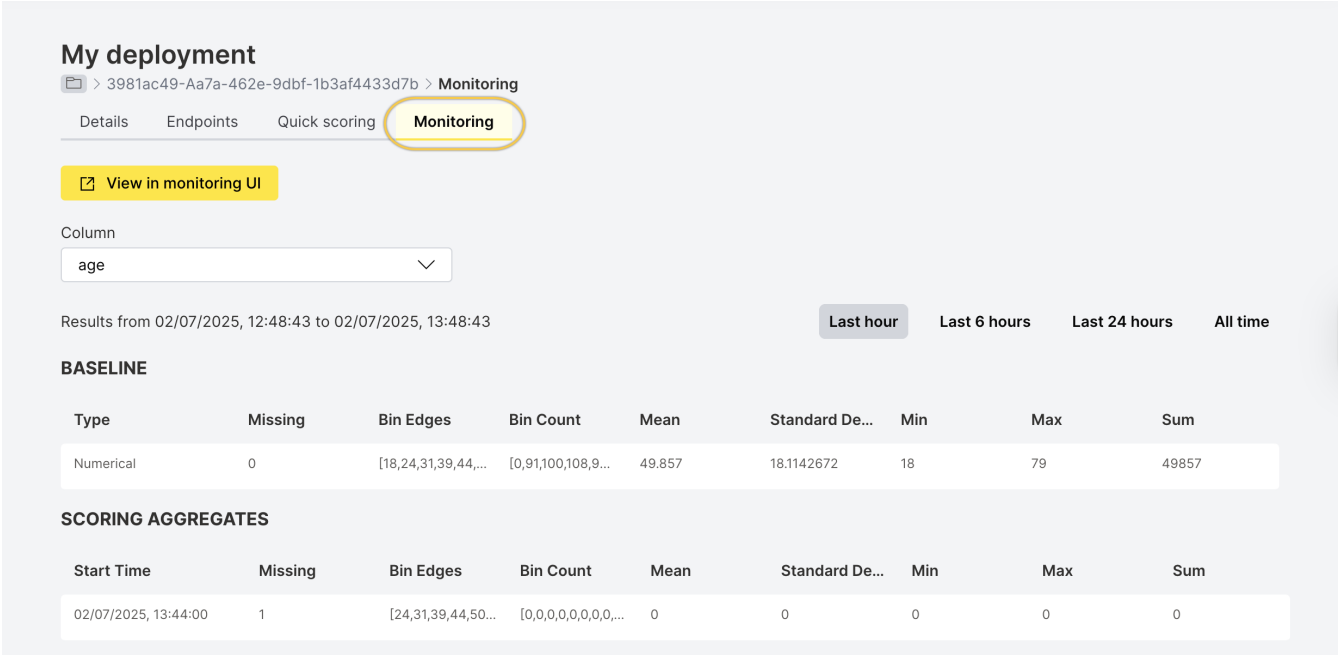


Step 4: View aggregated data

To view the scoring aggregates for each monitored column:

1. After scoring completes, go to the **Monitoring** tab.

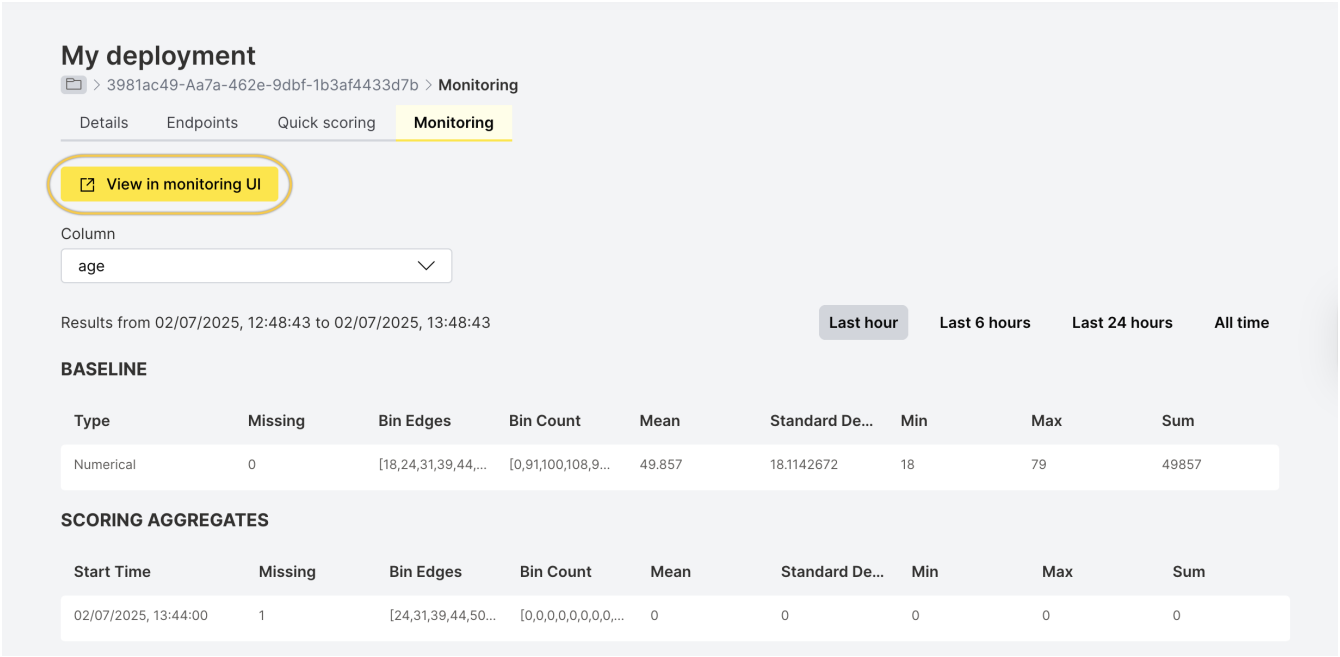
2. Wait 3–5 minutes to see data under **Scoring aggregates**.



Step 5: Analyze data in the monitoring UI

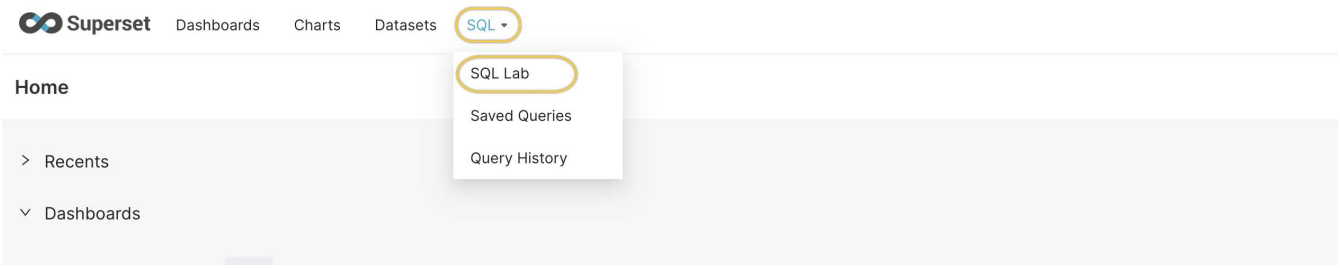
To view and analyze model drift:

1. Click **View in monitoring UI**.

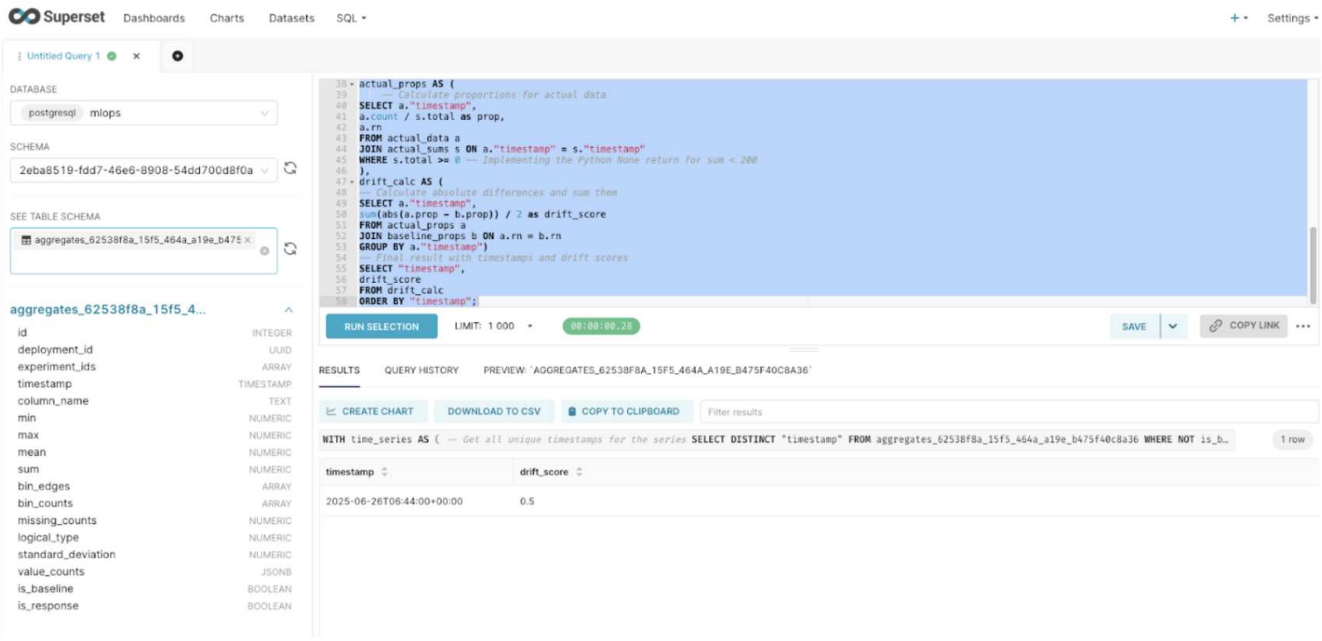


2. The Superset UI opens.

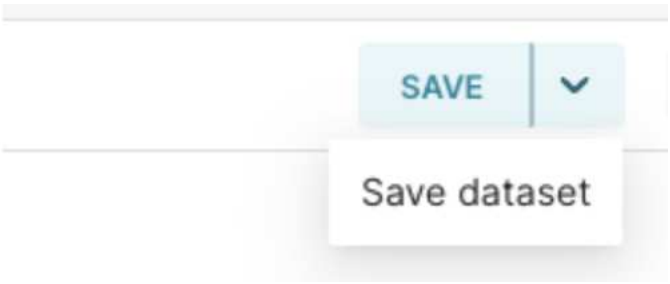
3. From the **SQL** drop-down, select **SQL Lab**.



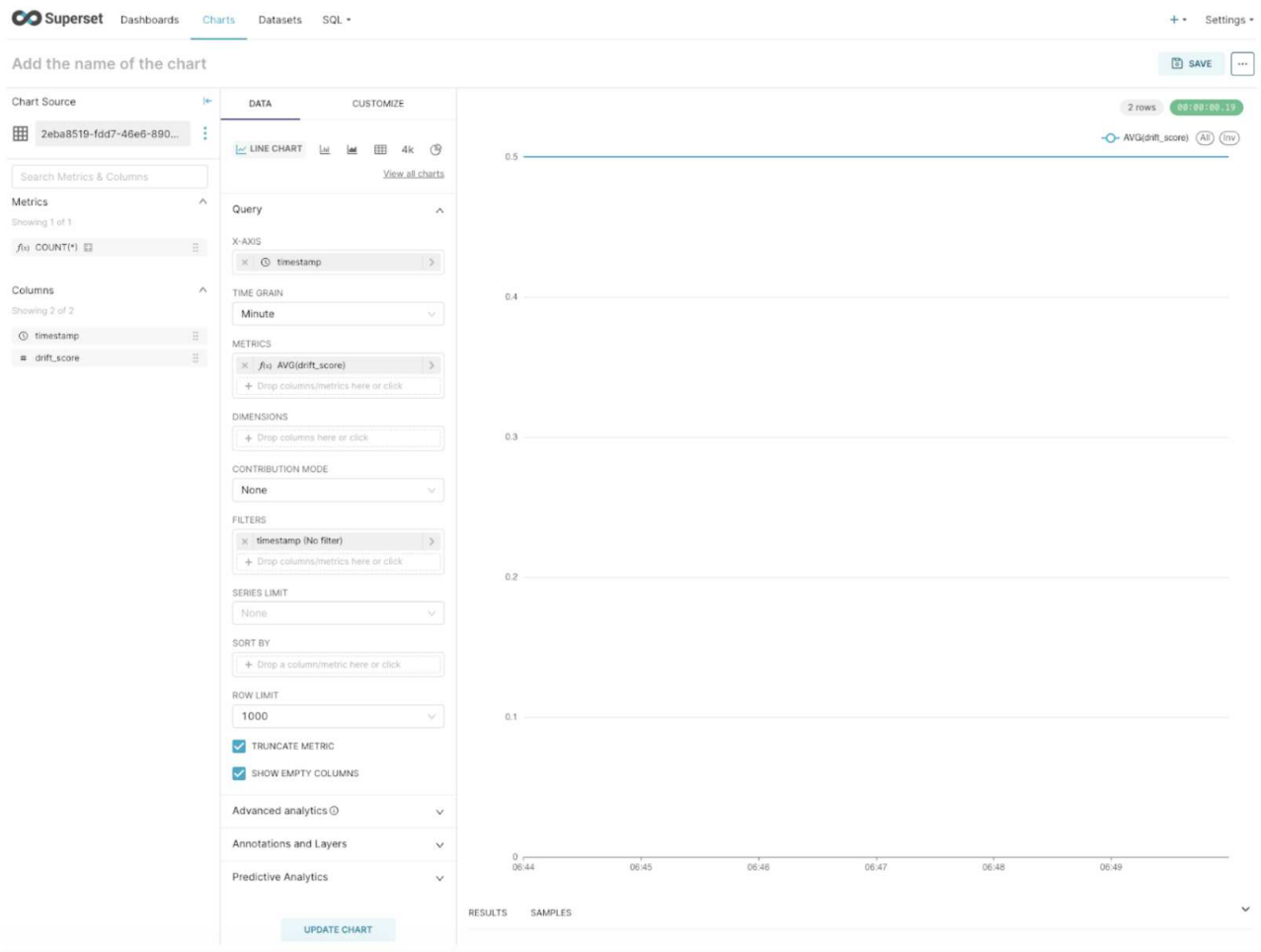
4. Enter the drift query:
Example:
```
sql
WITH time_series AS (
    -- Get all unique timestamps for the series
    SELECT DISTINCT "timestamp"
    FROM <TABLE_NAME>
    WHERE NOT is_baseline
    AND column_name = <FEATURE_NAME>
    ORDER BY "timestamp"),
baseline AS (
    -- Get the baseline (expected) data
    SELECT unnest(bin_counts) as count
    FROM <TABLE_NAME>
    WHERE is_baseline
    AND column_name = <FEATURE_NAME>),
baseline_sum AS (
    -- Calculate sum of baseline counts
    SELECT sum(count) as total
    FROM baseline),
baseline_props AS (
    -- Calculate baseline proportions
    SELECT count / total as prop,
    row_number() OVER () as rn
    FROM baseline,
    baseline_sum),
actual_data AS (
    -- Get actual data for each timestamp
    SELECT "timestamp",
    unnest(bin_counts) as count,
    row_number() OVER (PARTITION BY "timestamp") as rn
    FROM <TABLE_NAME>
    WHERE NOT is_baseline
    AND column_name = <FEATURE_NAME>),
actual_sums AS (
    -- Calculate sums for each timestamp
    SELECT "timestamp",
    sum(count) as total
    FROM actual_data
    GROUP BY "timestamp"),
actual_props AS (
    -- Calculate proportions for actual data
    SELECT a."timestamp",
    a.count / s.total as prop,
    a.rn
    FROM actual_data a
    JOIN actual_sums s ON a."timestamp" = s."timestamp"
    WHERE s.total >= 0 -- Implementing the Python None return for sum < 200
    ),
drift_calc AS (
    -- Calculate absolute differences and sum them
    SELECT a."timestamp",
    sum(abs(a.prop - b.prop)) / 2 as drift_score
    FROM actual_props a
    JOIN baseline_props b ON a.rn = b.rn
    GROUP BY a."timestamp")
    -- Final result with timestamps and drift scores
    SELECT "timestamp",
    drift_score
    FROM drift_calc
    ORDER BY "timestamp";
```



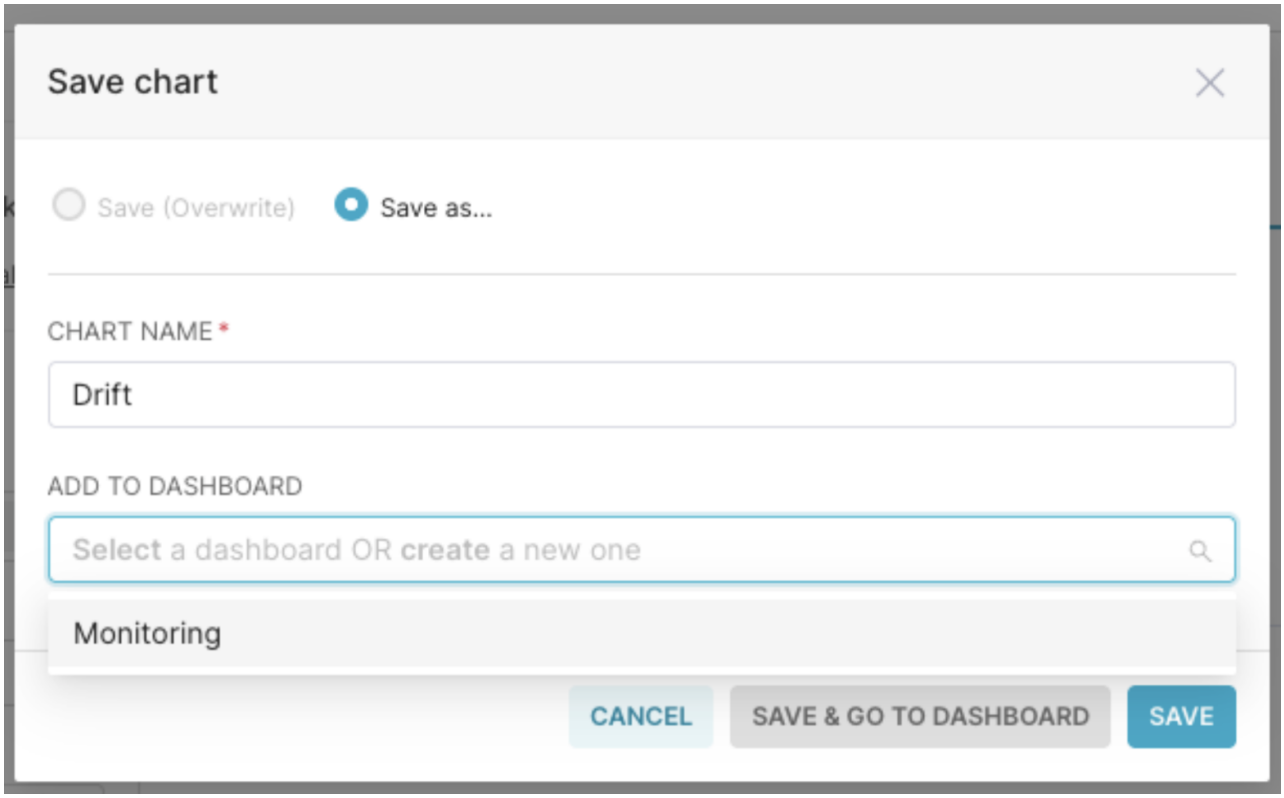
5. To save the result as a dataset, go to the **Save** drop-down and select **Save dataset**.



6. To convert it into a chart and add it to a dashboard:
    1. Go to **Charts** and customize the chart.

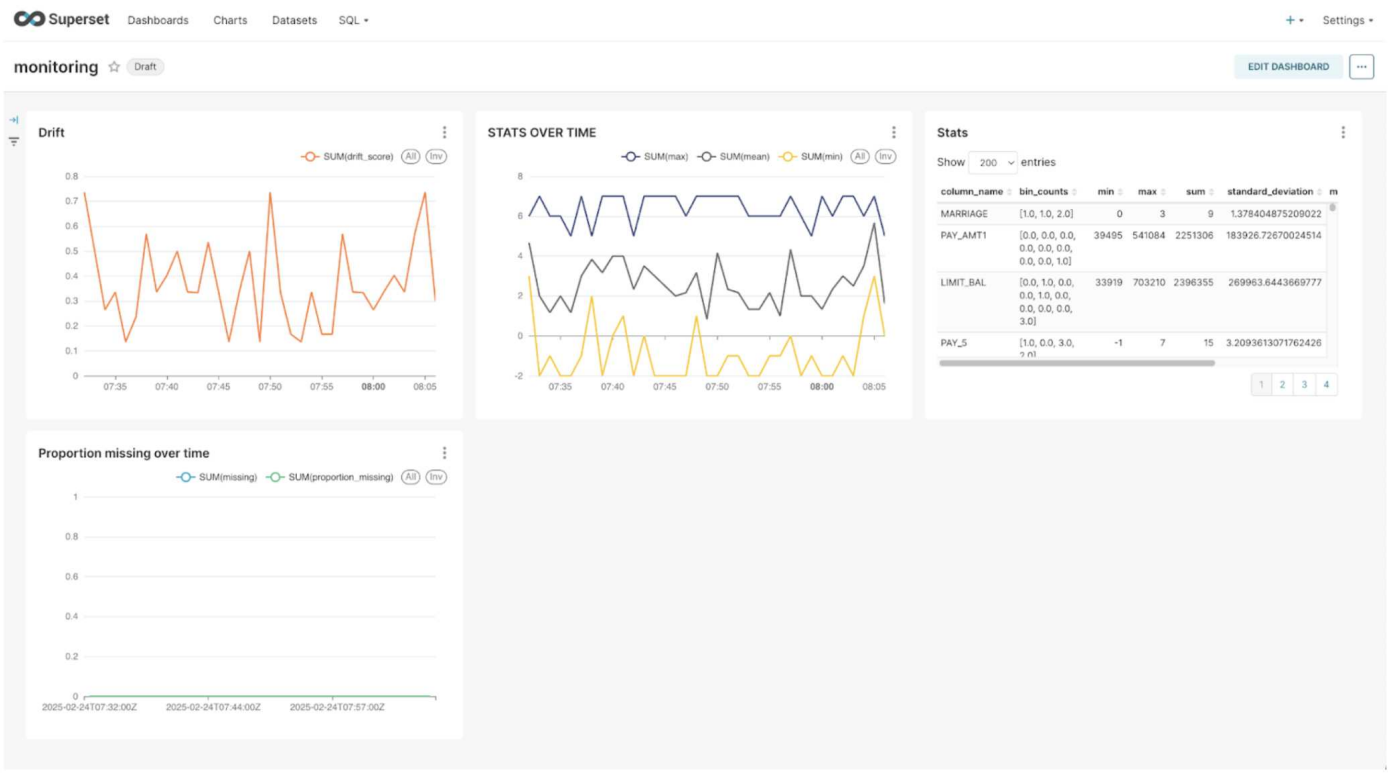


    2. Click **Save**, and add it to a dashboard.



3. Go to **Dashboards** and select the one you want to view.

You can explore more advanced dashboards for deeper insights.



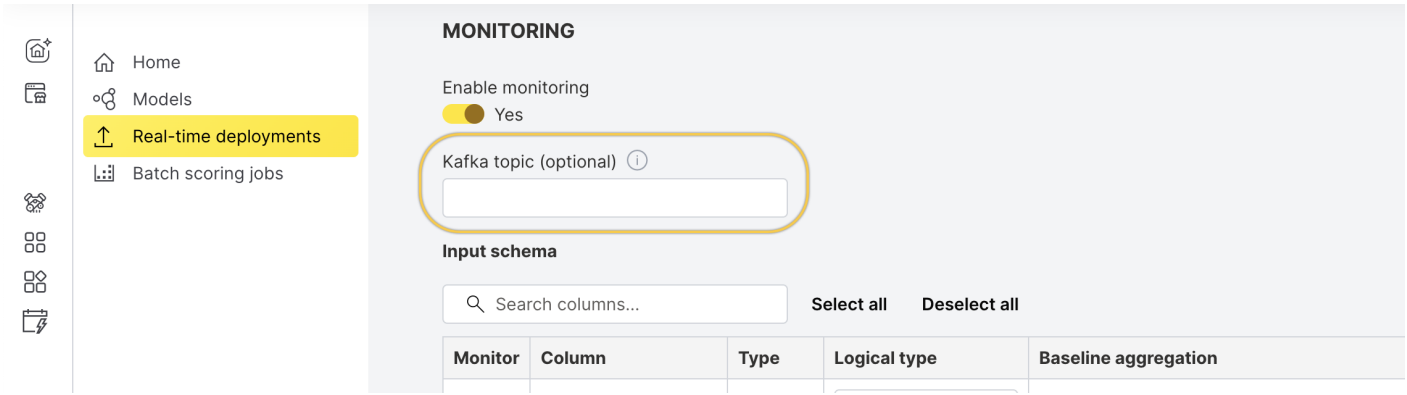
Configure model monitoring with the Python client

To learn how to configure monitoring for your deployment using the H2O MLOps Python client, see Monitoring setup.

Raw data export to Kafka

Monitoring supports exporting raw scoring data to Kafka. This feature allows users to process scoring data in ways required by their internal regulations. Request and response data from scoring operations can be sent to a specified Kafka topic for downstream processing, auditing, or debugging. This feature is enabled by the MLOPs administrator.

During model deployment with monitoring enabled, scoring data and response data are sent to a default topic configured by the MLOPs administrator. Users can optionally specify a custom Kafka topic where the data are sent for this particular deployment. This allows separating data streams per deployment for improved observability. This configuration has no effect in case the Kafka intgration is disabled by the admonistrator.



Note: The custom Kafka topic must exist before deploying the model. Monitoring will not attempt to create the topic automatically.

Once configured, the monitoring captures raw request and response data from scoring operations and forwards it to the configured Kafka topic (global or deployment-specific).

Common use cases for exporting raw data to Kafka include:

- Debugging and inspecting raw scoring payloads
- Auditing input/output for compliance
- Real-time analytics via stream processing systems

Batch scoring

Batch scoring is the process of making predictions on a large set of data all at once, instead of one-by-one in real time. This feature supports usage through both the UI and H2O MLOps Python client.

Batch scoring jobs in H2O MLOps create a dedicated Kubernetes runtime that reads data from an input source and stores the predicted results in an output location.

To run a batch scoring job, you must define the source of the input data and the location (sink) for the scored output.

H2O MLOps supports the following source and sink types:

- Azure Blob Storage
- Amazon S3
- Google Cloud Storage (GCS)
- MinIO
- JDBC

Note:

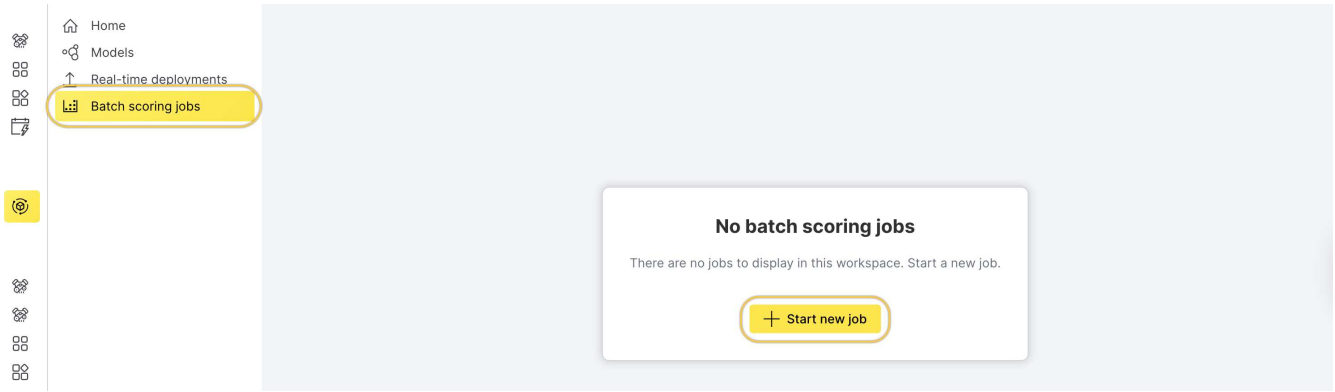
- **Supported input formats:** JDBC tables, CSV files with and without headers, and JSON files.
- **Supported output formats:** JDBC tables, CSV files without headers, and JSON files.

Batch scoring with the UI

This section describes how to start a batch scoring job using the H2O MLOps UI.

To batch score a model using the UI, follow these steps:

1. On the left navigation bar, click **Batch scoring jobs**.
2. Click **Start new job**.



3. On the **Start new job** page, enter a name for the batch scoring job in the **Job name** field.
4. Select the model from the **Model** drop-down menu.
5. Choose the artifact type and runtime from the **Artifact type and runtime** drop-down menu.
6. Under **Kubernetes options**, configure Kubernetes options, such as the number of replicas and resource requests and limits.
7. Under **Advanced settings**, configure the batch size.
8. Specify the source and sink configuration.

   Select the appropriate spec type (for example, **S3 Spec**) from the **Source spec** drop-down menu and fill out the configuration fields.

   Source spec

Note: The MinIO specification uses the same configuration fields as the S3 specification. To select MinIO as the source spec type, choose **S3 spec** from the source spec drop-down menu.

S3 spec

For **S3 Spec**, provide the following details:

- **accessKeyID** *(required)*: The unique identifier for AWS authentication. **Not required for public S3 buckets**.
- **secretAccessKey** *(required)*: The private password for AWS authentication. **Not required for public S3 buckets**.
- **sessionToken**: The temporary security token for time-limited access to AWS resources.
- **pathStyle**: Select this option to enable path-style URL construction for the S3 bucket.
- **region** *(required)*: The AWS geographical region where resources or services will be accessed.
- **endpoint**: The custom URL to override default AWS service endpoint for specialized configurations.
- **partSize**: The size of each partition in bytes for reading data.

Azure Blob Storage spec

For **Azure Blob Storage spec**, provide the following details:

- **accountKey**: The Azure storage account key. **or**
- **sasToken**: The shared access signature (SAS) token for accessing the storage account. **Note:**
- Either `accountKey` or `sasToken` is **required** to authenticate the source. You don't need to provide both.
- If you use a `sasToken`, make sure it includes `read`, `write`, and `list` permissions.
- **containerName** *(required)*: The name of the blob storage container.
- **partitionSize**: The size of each partition in bytes for reading data.

GCS spec

For **GCS spec**, provide the following details:

- **credentials** *(required)*: The service account JSON credentials.
- **projectID** *(required)*: The Google Cloud Project ID.
- **endpoint**: The custom endpoint URL.
- **partSize**: The size of each partition in bytes for reading data.

JDBC spec

For **JDBC spec**, provide the following details:

- **secretParams**: The set of key-value pairs that contain sensitive parameters (e.g., passwords) used to dynamically construct the JDBC connection string. Each key and value must be a string. For example, you can use `postgres://user:{{pass}}`, where `pass` is defined in `secretParams`.
- **driver** *(required)*: The JDBC driver to use. Supported values include `mysql`, `postgres`, `mssql`, and `oracle`.
- **table** *(required)*: The table to read from. You can also use any valid SQL expression for a `FROM` clause, such as a subquery enclosed in parentheses.
- **numPartitions**: The number of partitions to divide the table into for parallel reads. Required if partitioning is enabled. This setting determines the level of read parallelism.
- **lowerBound**: The lower boundary value used to compute partition strides. It is not used to filter rows and must match the data type of the `partitionColumn`.
- **upperBound**: The upper boundary value used to compute partition strides. Like `lowerBound`, this is only used for partitioning and must match the data type of the `partitionColumn`.
- **partitionColumn**: The column used to determine how the data is partitioned. This must be a numeric, date, or timestamp column.

    - **Source MIME type** *(required)*: The MIME type (media type) of the input data. Select an appropriate option from the drop-down menu.
    - **Source location** *(required)*: The path to the input data source.

    Now, select the appropriate spec type (for example, **S3 Spec**) from the **Sink spec** drop-down menu and fill out the configuration fields. ### Sink spec

Note: The MinIO specification uses the same configuration fields as the S3 specification. To select MinIO as the sink spec type, choose **S3 spec** from the sink spec drop-down menu.

S3 spec

For **S3 Spec**, provide the following details:

- **accessKeyID** *(required)*: The unique identifier for AWS authentication.
- **secretAccessKey** *(required)*: The private password for AWS authentication.
- **sessionToken**: The temporary security token for time-limited access to AWS resources.
- **pathStyle**: Select this option to enable path-style URL construction for the S3 bucket.
- **region** *(required)*: The AWS geographical region where resources or services will be accessed.
- **endpoint**: The custom URL to override default AWS service endpoint for specialized configurations.
- **writeConcurrency**: The number of concurrent write operations.

Azure Blob Storage spec

For **Azure Blob Storage spec**, provide the following details:

- **accountKey**: The Azure storage account key. **or**
- **sasToken**: The shared access signature (SAS) token for accessing the storage account. **Note:**
- Either `accountKey` or `sasToken` is **required** to authenticate the sink. You don't need to provide both.
- If you use a `sasToken`, make sure it includes `read`, `write`, and `list` permissions.
- **containerName** *(required)*: The name of the blob storage container.
- **writeConcurrency**: The number of concurrent write operations allowed.

GCS spec

For **GCS spec**, provide the following details:

- **credentials** *(required)*: The service account JSON credentials.
- **projectID** *(required)*: The Google Cloud Project ID.
- **endpoint**: The custom endpoint URL.
- **writeConcurrency**: The number of concurrent write operations.

JDBC spec

For **JDBC spec**, provide the following details:

- **secretParams**: The set of key-value pairs that contain sensitive parameters (e.g., passwords) used to dynamically construct the JDBC connection string. Each key and value must be a string. For example, you can use `postgres://user:{{pass}}`, where `pass` is defined in `secretParams`.
- **driver** *(required)*: The JDBC driver to use. Supported values include `mysql`, `postgres`, `mssql`, and `oracle`.
- **table** *(required)*: The JDBC table that should be write into.
    - **Sink MIME type** *(required)*: The MIME type (media type) of the output data. Select an appropriate option from the drop-down menu.
    - **Sink location** *(required)*: The destination path where the output data will be written.

9. After filling out the configuration fields, click **Start job** to initiate the batch scoring job.

Batch scoring with Python client

To learn how to perform batch scoring using the H2O MLOps Python client, see the Batch scoring example in the Python client examples section.

H2O MLOps Python client

The H2O MLOps Python client lets you use the H2O MLOps API from your Python application. This guide describes how you can install the H2O MLOps Python client, connect to H2O MLOps and carry out tasks using the Python client. After successful installation, you can interact with the H2O MLOps API via the H2O MLOPs gRPC Gateway. For more information, see H2O MLOps gRPC Gateway.

The H2O MLOps Python client documentatin is organized into the following sections:

- Installation >The Python client can be easily installed to get started with H2O MLOps programmatically. Follow our installation guide to set up the client in your environment and prepare for integration with the platform.
- Getting started >The Python client makes it easy to connect to H2O MLOps, deploy models, and score data from your Python code. Follow our quickstart guide for an end-to-end workflow to create a workspace, register a model, deploy it, and score against the deployment.
- Examples

  This section provides code examples for performing common operations using the H2O MLOps Python client. Each example shows how to perform specific tasks in your workflow.

  - Connect to H2O MLOps >Learn recommended methods to establish a secure connection to the H2O MLOps using the Python client.
  - Manage Workspaces > Discover how to create, manage, and organize your workspaces in H2O MLOps using the Python client with better collaboration and resource management.
  - Manage Experiments >Track and manage your machine learning experiments using H2O MLOps Python client, including experiment tags. It also describes experiment properties and how to compute Kubernetes options.
  - Handle artifacts >Learn how to use the Python client to add, retrieve, update, delete, or convert artifacts to strings or dictionaries for an H2O MLOps entity.
  - Manage Models >Explore comprehensive model management capabilities, including versioning through code.
  - Deploy Models
    - Configure deployments > Explore the available options for configuring deployments using the H2O MLOps Python client.
    - Manage deployments > Learn how to create, view, update, and delete deployments, as well as how to view logs and configure endpoints using the H2O MLOps Python client.
    - Deployment scorer > Learn how to use the H2O MLOps Python client to score against deployments.
  - Batch scoring >Implement efficient batch prediction workflows for processing large volumes of data with your deployed models.
  - Monitoring setup >Configure comprehensive monitoring for your deployed models to track performance, data drift, and operational metrics.

Installation

To install the H2O MLOps Python client, run the following command:

```
pip install h2o-mlops
```

Note: The minimum Python client version required for this MLOps release is v1.4.0.

To install a specific version of the H2O MLOps Python client, run:

```
pip install h2o-mlops==x.y.z
```

To install the H2O MLOps Python client with a minimum version requirement, run:

```
pip install h2o-mlops>=x.y.z
```

Note: The minimum Python version required for the latest client is Python 3.9.

Version compatibility

note Python Client version compatibility

Starting with H2O MLOps v1.0.0, use Python client version v1.4.0 or later. Newer Python client versions are not backward compatible with MLOps versions earlier than v1.0.0.

From v1.0.0 onwards, you can use the latest available Python client, as backward compatibility is maintained. However, upgrading or downgrading may require minor code changes due to potential breaking changes. For details, see the migration guide.

For H2O MLOps versions from v0.66.1 to v0.70.5, use Python client version v1.3.3.

For versions earlier than v0.66.1, it is recommended to use the corresponding client version that matches the H2O MLOps release. For example, client version 0.62.1a5 is intended for use with H2O MLOps version 0.62.1. Using a client with a different version of H2O MLOps may lead to compatibility issues.

Getting started

This page helps you get started with the H2O MLOps Python client by walking through a complete quickstart workflow, from connecting to H2O MLOps, to deploying a model, and scoring data against the deployment.

H2O MLOps enables teams to manage the lifecycle of machine learning models, including registration, deployment, monitoring, and scoring. The Python client allows you to perform these tasks directly from your Python code.

Follow these steps to connect to your H2O MLOps environment, create a workspace, register a model, deploy it, and score data against the deployment.

Prerequisites

Before you begin, install the H2O MLOps Python client. For more information, see Installation.

Step 1: Import the required packages

```
import h2o_mlops
import h2o_mlops.options as options
import h2o_mlops.types as types
```

Step 2: Initialize the H2O MLOps client

Connect to H2O MLOps using your H2O Cloud URL, refresh token, and SSL certificate (if required):

```
mlops = h2o_mlops.Client(
   h2o_cloud_url=<H2O_CLOUD_URL>,
   refresh_token=<REFRESH_TOKEN>,
   ssl_cacert="/path/to/your/ca_certificate.pem",
)
```

Note: Replace placeholders with your actual credentials and file paths. If SSL is not needed, you can omit `ssl_cacert`.

Step 3: Create a workspace

Workspaces are containers that group related models, deployments, and artifacts. Create a new one:

```
workspace = mlops.workspaces.create(name="my-workspace")
```

Step 4: Register an experiment as a model version

Register an existing experiment artifact as a model version to make it available for deployment:

```
model = workspace.models.register(
   experiment="/path/to/my_experiment_artifact.zip",
   name="my-experiment",
)
```

Make sure the path points to a valid model artifact on your local machine.

Step 5: Deploy the model

Deploy the registered model as an API endpoint using a supported scoring runtime:

```
deployment = workspace.deployments.create(
   name="my-deployment",
   composition_options=options.CompositionOptions(
       model=model,
       scoring_runtime=model.experiment().scoring_runtimes[0]
   ),
   security_options=options.SecurityOptions(
       security_type=types.SecurityType.DISABLED,
   ),
)
```

Note: Make sure to use the index that matches the scoring runtime you want from `model.experiment().scoring_runtimes`.

Step 6: Wait for the deployment to become healthy

Deployment may take a few seconds. Use the following to wait until it's ready:

```
deployment.wait_for_healthy()
```

Step 7: Score data against the deployment

Once the deployment is healthy, you can send data to it for scoring:

```
deployment.scorer.score(
   payload={
       "fields": [
           "Origin", "Dest", "fDayofMonth", "fYear", "UniqueCarrier", "fDayOfWeek", "fMonth", "IsDepDela
       ],
       "rows": [
           ["text", "text", "text", "text", "text", "text", "text", "text"],
           ["text", "text", "text", "text", "text", "text", "text", "text"],
           ["text", "text", "text", "text", "text", "text", "text", "text"],
       ]
   },
)
```

The output contains predictions for the provided input rows:

```
{'id': '2afe0ab6-db1c-4ecc-abb2-747340b3b8dc',
'fields': ['Distance'],
'score': [['713.7770420135266'],
 ['713.7770420135266'],
 ['713.7770420135266']]}
```

Explore more examples

This **Getting started** page covered the basics of connecting to H2O MLOps, deploying a model, and scoring data using the H2O MLOps Python client.

To learn more, see the **Examples** section, which includes code examples for the following operations:

- Connect to H2O MLOps
- Manage Workspaces
- Manage Experiments
- Handle artifacts
- Manage Models
- Configure deployments
- Manage deployments
- Deployment scorer
- Batch scoring
- Monitoring setup

Connect to H2O MLOps

This page describes recommended methods for connecting to H2O MLOps with the Python Client. Select one of the following methods based on how your H2O Cloud is set up and where you are connecting from.

1. Connect with SSL verification enabled
2. Connect with private certificate
3. Connect with SSL verification disabled
4. Connect from H2O Notebook Labs

Prerequisites

Before you connect to H2O MLOps, make sure you complete the following steps.

1. Import the necessary Python packages. For instructions, see Step 1: Import the required packages.
2. Gather required values. You need two values to connect:
   - `h2o_cloud_url`: The URL used to access the H2O Cloud homepage.
   - `refresh_token`: Obtain the `refresh_token` from the H2O Cloud UI. For instructions, see Get the platform token.

The H2O MLOps Python Client also supports the following optional SSL settings:

- `verify_ssl`: If set to `True` (the default value), the client will check that the server's SSL certificate is valid.
- `ssl_cacert`: A path to a custom CA (Certificate Authority) certificate or bundle in `.pem`, or `.crt` format.

The Python client will also check environment variables and automatically use them if no arguments are supplied:

- `h2o_cloud_url`: `H2O_CLOUD_ENVIRONMENT`
- `refresh_token`: `H2O_CLOUD_CLIENT_PLATFORM_TOKEN`
- `ssl_cacert`: `MLOPS_AUTH_CA_FILE_OVERRIDE`

You can connect to the H2O MLOps client with SSL verification either enabled or disabled. Use one of the following examples based on your SSL configuration preferences.

Connect with SSL verification enabled

Connection with SSL verification is a default mode for MLOPs Python Client.

```
import h2o_mlops

mlops = h2o_mlops.Client(
   h2o_cloud_url=...,
   refresh_token=...,
)
```

Connect with private certificate

To connect to an environment that uses a private certificate, please follow the following:

```
import h2o_mlops

mlops = h2o_mlops.Client(
   h2o_cloud_url=...,
   refresh_token=...,
   ssl_cacert="/path/to/your/ca_certificate.pem",
)
```

Connect with SSL verification disabled

```
import h2o_mlops

mlops = h2o_mlops.Client(
   h2o_cloud_url=...,
```

```
    refresh_token=...,
    verify_ssl=False,
)
```

Connect from H2O Notebook Labs

H2O Notebook Labs in the H2O Cloud will detect your user and automatically connect to H2O MLOps. For more information on H2O Notebook Labs, see the H2O Notebook Labs documentation.

Input:

```
import h2o_mlops

mlops = h2o_mlops.Client()
```

Verify the connection

After establishing the connection, verify it by checking your user information.

Input:

```
mlops.users.get_me()
```

Output:

```
<class 'h2o_mlops._users.MLOpsUser(
    uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
    username='user',
    name='User',
    email='user@h2o.ai',
)'>
```

Advanced configurations

Configurable timeout settings

You can configure the following timeout parameters when you connect to H2O MLOps:

- `global_request_timeout: Optional[float | Tuple[float, float]] = None` – Specify the timeout for general API requests in seconds.
- `file_transfer_timeout: Optional[float | Tuple[float, float]] = None` – Specify the timeout for file upload and download requests in seconds.

If a single number is provided, it represents the total timeout for the respective operation. Alternatively, a pair (tuple) of (connection, read) timeouts can be specified.

Example:

```
import h2o_mlops

mlops = h2o_mlops.Client(
    h2o_cloud_url=...,
    refresh_token=...,
    global_request_timeout=(5, 15),
    file_transfer_timeout=40,
)
```

Manage Workspaces

This guide explains how to create, view, update, and delete workspaces in H2O using the H2O MLOps Python client.

To learn more about workspaces, see Workspaces.

Prerequisites

Before you begin,

- Connect to H2O MLOps. For instructions, see Connect to H2O MLOps.

Create a workspace

This section describes how to create a new workspace in H2O using the H2O MLOps Python client.

Create a new workspace using the `create()` method.

```
workspace = mlops.workspaces.create(name="my-workspace", description="my-workspace")
```

Parameters:

- `name`: The name of the workspace.
- `description`: A short description of the workspace.

View workspaces

This section describes how to view existing workspaces in H2O using the H2O MLOps Python client.

Count workspaces

Get the total number of workspaces:

Input:

```
mlops.workspaces.count()
```

Output:

```
4
```

List all workspaces

List all existing workspaces using the `list()` method.

Input:

```
workspaces = mlops.workspaces.list()
workspaces
```

This returns a list of all available workspaces.

Output:

```
>python >    | name                | uid >---+--------------------
+------------------------------------ >  0 | my-workspace       | 7bdc6a96-804e-452a-b7dd-
8afc1968b3d9 >  1 | dummy              | 559140a8-34de-48f1-ab55-27d805d2f197 >  2 | Personal
Workspace | 92fb7ec4-a011-46b1-bff4-4669d9ab17ee >  3 | Appstore Workspace | appstore >
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(workspaces)` returns the total number of rows it contains, not just the number currently displayed.
- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: `>python >workspaces.show(n=100) >` This will display up to 100 workspaces.
- The `workspaces` can be iterated over, as it is designed to behave like an iterator.

List workspace aggregates

Use the `aggregates()` method to view workspaces along with the number of models, model versions, and attached experiments.

Input:

```
mlops.workspaces.aggregates()
```

Output:

```
   | name                |   versions |   models |   experiments | uid
---+---------------------+------------+----------+---------------+--------------------------------------
 0 | my-workspace        |          0 |        0 |             0 | 7bdc6a96-804e-452a-b7dd-8afc1968b3d9
 1 | dummy               |          2 |        1 |             2 | 559140a8-34de-48f1-ab55-27d805d2f197
 2 | Personal Workspace  |          0 |        0 |             0 | 92fb7ec4-a011-46b1-bff4-4669d9ab17ee
```

Filter workspaces

Use the `list()` method with key-value arguments to filter the workspaces.

Input:

```
mlops.workspaces.list(name="my-workspace")
```

This returns a list of matching workspaces as a table.

Output: >python > | name | uid >---+---+---------- > 0 | my-workspace | 7bdc6a96-804e-452a-b7dd-8afc1968b3d9 >

Retrieve a workspace

To retrieve a specific workspace, use the `get()` method with the workspace UID.

Input:

```
workspace = mlops.workspaces.get(uid="7bdc6a96-804e-452a-b7dd-8afc1968b3d9")
workspace
```

Note: You can also retrieve a specific workspace from the list returned by `list()` using indexing.
For example, `workspace = mlops.workspaces.list(key=value)[index]`. The `key` and `value` arguments are optional.

Output:

```
<class 'h2o_mlops._workspaces.Workspace(
   uid='7bdc6a96-804e-452a-b7dd-8afc1968b3d9',
   name='my-workspace',
   description='my-workspace',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 7, 17, 15, 43, 0, 583925, tzinfo=tzutc()),
   last_modified_time=None,
)'>
```

Workspace properties

A workspace has the following main properties:

- `uid`: The unique identifier of the workspace.
- `name`: The name of the workspace.
- `description`: A description of the workspace.
- `creator`: The user who created the workspace.
- `created_time`: The timestamp when the workspace was created.
- `last_modified_time`: The timestamp of the last modification.
- `last_modified_by`: The user who made the last modification.

Aggregate

You can use `workspace.aggregate` to retrieve the number of:

- Versions
- Models
- Experiments

Input:

```
workspace.aggregate
```

Output:

```
    | entity      |   count
---+-------------+---------
 0 | versions    |       0
 1 | models      |       0
 2 | experiments |       0
```

Update a workspace

You can update only the `name` and `description` fields of a workspace.

Make sure to retrieve the workspace instance as described in Retrieve a workspace before proceeding.

Input:

```
workspace.update(name="my-new-workspace")
workspace
```

Output:

```
<class 'h2o_mlops._workspaces.Workspace(
   uid='7bdc6a96-804e-452a-b7dd-8afc1968b3d9',
   name='my-new-workspace',
   description='my-workspace',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 7, 17, 15, 43, 0, 583925, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 7, 17, 15, 43, 4, 696432, tzinfo=tzutc()),
)'>
```

Delete a workspace

danger warning Deleting a workspace also deletes all entities within it.

To delete a workspace, use the `delete()` method:

Input:

```
workspace.delete()
```

Manage Experiments

This page explains how to create, view, update, and delete experiments; add comments; and manage experiment tags in H2O MLOps using the Python client. It also describes experiment properties and how to compute Kubernetes options.

To learn more about experiments, see Experiments.

Prerequisites

Before you begin, 1. Connect to H2O MLOps. For instructions, see Connect to H2O MLOps. 2. Create a workspace. For steps, see Create a workspace.

Create an experiment

Use the `create()` method to create a new experiment in a workspace:

```
experiment = workspace.experiments.create(
   data="/path/test.zip",
   name="my-experiment",
   description="Test experiment",
)
```

Note: You can link or unlink an H2O Driverless AI (DAI) experiment, or an existing DAI or H2O MLOps experiment in storage, to a workspace.

Link an experiment by UID:

```
workspace.experiments.link(uid="your-experiment-uid")
```

Unlink an experiment:

```
workspace.experiments.unlink(uid="your-experiment-uid")
```

View experiments

Count experiments

Get the total number of experiments in a workspace:

Input:

```
workspace.experiments.count()
```

Output:

```
1
```

List experiments

List all experiments in a workspace:

Input:

```
experiments = workspace.experiments.list()
experiments
```

Output:

```
   | name          | uid                                  | tags
---+---------------+--------------------------------------+--------
 0 | my-experiment | d9a47c99-c66c-4ff9-b2b6-30faf5f413ef |
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(experiments)` returns the total number of rows it contains, not just the number currently displayed.

- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: >`python >experiments.show(n=100) >` This will display up to 100 experiments.
- The `experiments` can be iterated over, as it is designed to behave like an iterator.

Filter experiments

Use the `list()` method with key-value arguments to filter the experiments.

Input:

```
workspace.experiments.list(name="my-experiment")
```

This returns a list of matching experiments as a table.

Output:

```
   | name                 | uid                                  | tags
---+----------------------+--------------------------------------+--------
 0 | my-experiment        | d9a47c99-c66c-4ff9-b2b6-30faf5f413ef |
```

Retrieve an experiment

Retrieve a specific experiment by UID:

Input:

```
experiment = workspace.experiments.get(uid="d9a47c99-c66c-4ff9-b2b6-30faf5f413ef")
experiment
```

Note: You can also retrieve a specific experiment from the list returned by `list()` using indexing.
For example, `experiment = workspace.experiments.list(key=value)[index]`. The `key` and `value` arguments are optional.

Output:

```
<class 'h2o_mlops._experiments.MLOpsExperiment(
   uid='d9a47c99-c66c-4ff9-b2b6-30faf5f413ef',
   name='my-experiment',
   description='Test experiment',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 5, 22, 7, 2, 48, 185159, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 5, 22, 7, 2, 48, 185159, tzinfo=tzutc()),
)'>
```

Experiment properties

An experiment has the following main properties:

- `uid`: The unique identifier of the experiment.
- `name`: The name of the experiment.
- `description`: A description of the experiment.
- `creator`: The user who created the experiment.
- `created_time`: The timestamp when the experiment was created.
- `last_modified_time`: The timestamp of the last modification.
- `is_registered`: If the experiment is registered or not.

Metadata

Each experiment includes metadata you can retrieve using the following method:

Input:

```
experiment.metadata
```

Output:

```
   | key                | value
---+--------------------+-------------------------------------------
 0 | h2o3/algo          | glm
 1 | h2o3/algo_full_name | Generalized Linear Modeling
 2 | h2o3/category      | Regression
 3 | h2o3/columns       | ['Origin', 'Dest', 'fDayofMonth', 'fYear...
 4 | h2o3/created_time  | 2020-08-24 07:21:50.137000+00:00
 5 | input_schema       | [{'name': 'Origin', 'type': 'STR'}, {'na...
 6 | model_type         | h2o3/mojo
 7 | output_schema      | [{'name': 'Distance', 'type': 'FLOAT64'}...
 8 | tool               | h2o3
```

To get a specific metadata entry by index:

Input:

```
experiment.metadata[3]
```

Output:

```
{'h2o3/columns': ['Origin',
 'Dest',
 'fDayofMonth',
 'fYear',
 'UniqueCarrier',
 'fDayOfWeek',
 'fMonth',
 'IsDepDelayed']}
```

Parameters

To access parameters related to the dataset used in the experiment:

Input:

```
experiment.parameters["target_column"]
```

Output:

```
{'training_dataset_id': '',
'validation_dataset_id': '',
'test_dataset_id': '',
'target_column': 'Distance',
'weight_column': '',
'fold_column': ''}
```

In this example, the target column is `Distance`.

Statistics

To access training statistics:

Input:

```
experiment.statistics
```

Output:

```
{'training_duration': None}
```

Input schema

To view the schema of the input dataset:

Input:

```
experiment.input_schema
```

Output:

```
   | name          | type
---+---------------+--------
 0 | Origin        | STR
 1 | Dest          | STR
 2 | fDayofMonth   | STR
 3 | fYear         | STR
 4 | UniqueCarrier | STR
 5 | fDayOfWeek    | STR
 6 | fMonth        | STR
 7 | IsDepDelayed  | STR
```

Output schema

To view the output schema of the experiment:

Input:

```
experiment.output_schema
```

Output:

```
   | name     | type
---+----------+---------
 0 | Distance | FLOAT64
```

Scoring runtimes

You can list the available scoring runtimes that might be used when deploying the experiment:

Input:

```
scoring_runtimes = experiment.scoring_runtimes
scoring_runtimes
```

Output:

```
   | artifact_type   | runtime_uid                           | runtime_name
---+-----------------+---------------------------------------+-----------------------------------------
 0 | h2o3_mojo       | h2o3_mojo_runtime                     | H2O-3 MOJO Scorer
 1 | h2o3_mojo       | h2o3_mojo_runtime_shapley_transformed | H2O-3 MOJO Scorer (Shapley transformed on
```

To view details of a specific scoring runtime:

Input:

```
scoring_runtimes[0]
```

Output:

```
<class 'h2o_mlops._runtimes.MLOpsScoringRuntime(
   runtime='h2o3_mojo_runtime',
   artifact_type='h2o3_mojo',
   artifact_processor='h2o3_mojo_extractor',
   model_type='h2o3_mojo',
)'>
```

Compute Kubernetes options

To compute Kubernetes options for an experiment runtime:

Input:

```
experiment.compute_k8s_options(
   runtime_uid="h2o3_mojo_runtime", workers=1
)
```

Output:

```
KubernetesOptions(
   replicas=1,
   requests={'cpu': '500m', 'memory': '128Mi'},
   limits={},
   affinity=None,
   toleration=None
)
```

Update an experiment

You can update only the `name` and `description` fields of an experiment.

Make sure to retrieve the experiment instance before updating it. See Retrieve an experiment.

Input:

```
experiment.update(name="new-experiment")
experiment
```

Output:

```
<class 'h2o_mlops._experiments.MLOpsExperiment(
   uid='d9a47c99-c66c-4ff9-b2b6-30faf5f413ef',
   name='new-experiment',
   description='Test experiment',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 5, 22, 7, 2, 48, 185159, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 5, 22, 7, 3, 26, 278369, tzinfo=tzutc()),
)'>
```

Add comments to an experiment

You can add one or more comments to an experiment to share information with collaborators.

```
experiment.comments.add("Comment 01")
experiment.comments.add("Comment 02")
```

To list all comments:

Input:

```
experiment.comments.list()
```

Output:

```
   | created_time              | author_username     | message
---+--------------------------+---------------------+------------
 0 | 2025-05-22 07:03:30 AM | user                | Comment 01
 1 | 2025-05-22 07:03:31 AM | user                | Comment 02
```

Manage experiment tags

You can create tags and add them to experiments to group related experiments. For example, you can create a tag called `Telco` for telecommunication-related experiments and later retrieve them as a group.

Create a tag

To create a new tag in a workspace:

```
tag1 = workspace.tags.create(label="tag1")
```

List tags

To list all tags in a workspace:

Input:

```
workspace.tags.list()
```

Output:

```
   | label    | uid
---+----------+--------------------------------------
 0 | tag1     | 31fc1901-1134-4384-ac07-e0965f8e30c7
```

Get a specific tag

To retrieve a tag by its label:

Input:

```
tag1 = workspace.tags.get(label="tag1")
tag1
```

Output:

```
<class 'h2o_mlops._projects.MLOpsProjectTag(
   uid='31fc1901-1134-4384-ac07-e0965f8e30c7',
   label='tag1',
   parent_workspace_uid='e37a6146-5248-4754-9d93-68a9798babb2',
   created_time=datetime.datetime(2025, 5, 22, 7, 3, 35, 596458, tzinfo=tzutc()),
)'>
```

Add tag

To add an existing tag to an experiment:

```
experiment.tags.add(label="tag1")
```

To add a new tag to an experiment without creating it first:

```
experiment.tags.add(label="tag2")
```

To list and verify all tags in an experiment:

Input:

```
experiment.tags.list()
```

Output:

```
   | label    | uid
---+----------+--------------------------------------
 0 | tag1     | 31fc1901-1134-4384-ac07-e0965f8e30c7
 1 | tag2     | b541b0b1-6371-4106-8095-94c9c25f2f0a
```

Update a tag

To update a tag's label and view the updated list:

Input:

```
tag1.update(label="new-tag1")
experiment.tags.list() # or workspace.tags.list()
```

Output:

```
   | label     | uid
---+-----------+--------------------------------------
 0 | tag2      | b541b0b1-6371-4106-8095-94c9c25f2f0a
 1 | new-tag1  | 31fc1901-1134-4384-ac07-e0965f8e30c7
```

Remove a tag

To remove a tag from an experiment:

Input:

```
experiment.tags.remove(label="new-tag1")
experiment.tags.list()
```

Output:

```
   | label   | uid
---+---------+-------------------------------------
 0 | tag2    | b541b0b1-6371-4106-8095-94c9c25f2f0a
```

This removes the tag only from the experiment, not from the workspace.

To view all tags in a workspace after performing the above removal, use the following method:

Input:

```
workspace.tags.list()
```

This displays the list of tags currently available in the workspace.

Output:

```
   | label    | uid
---+----------+-------------------------------------
 0 | tag2     | b541b0b1-6371-4106-8095-94c9c25f2f0a
 1 | new-tag1 | 31fc1901-1134-4384-ac07-e0965f8e30c7
```

Delete a tag

To delete a tag from a workspace:

Input:

```
tag1.delete()
workspace.tags.list()
```

Output:

```
   | label   | uid
---+---------+-------------------------------------
 0 | tag2    | b541b0b1-6371-4106-8095-94c9c25f2f0a
```

Note: You cannot delete a tag from a workspace if it has already been added to an experiment.

Delete and restore experiments

This section describes how to delete and restore experiments.

Delete using an experiment instance

If you already have a reference to the `experiment` object, use the `delete()` method:

Input:

```
experiment.delete()
workspace.experiments.list()
```

Output:

```
   | name   | uid   | tags
---+--------+-------+--------
```

Restore using an experiment instance

If you already have a reference to the `experiment` object, use the `restore()` method:

Input:

```
experiment.restore()
workspace.experiments.list()
```

Output:

```
   | name                  | uid                                  | tags
---+-----------------------+--------------------------------------+--------
 0 | new-experiment        | d9a47c99-c66c-4ff9-b2b6-30faf5f413ef | tag2
```

Delete using experiment UIDs

You can also delete multiple experiments at once by specifying their UIDs:

Input:

```
workspace.experiments.delete(uids=["d9a47c99-c66c-4ff9-b2b6-30faf5f413ef"])
```

Note: You can also pass a list of `MLOpsExperiment` instances or a `_utils.Table` containing experiments. Example: `workspace.experiments.delete(experiments=[experiment])`

Output:

```
   | experiment_uid                       | is_deleted   | message   | workspace_uid
---+--------------------------------------+--------------+-----------+----------------------------------
 0 | d9a47c99-c66c-4ff9-b2b6-30faf5f413ef | True         |           | e37a6146-5248-4754-9d93-68a9798ba
```

Restore using experiment UIDs

You can also restore multiple experiments at once by specifying their UIDs:

Input:

```
workspace.experiments.restore(uids=["d9a47c99-c66c-4ff9-b2b6-30faf5f413ef"])
```

Note: You can also pass a list of `MLOpsExperiment` instances or a `_utils.Table` containing experiments. Example: `workspace.experiments.restore(experiments=[experiment])`

Output:

```
   | experiment_uid                       | is_restored   | message   | workspace_uid
---+--------------------------------------+---------------+-----------+----------------------------------
 0 | d9a47c99-c66c-4ff9-b2b6-30faf5f413ef | True          |           | e37a6146-5248-4754-9d93-68a9798b
```

Handle artifacts

This guide explains how to add artifacts to an MLOps entity, such as a workspace, dataset, experiment, or deployment, as well as how to retrieve, update, delete, or convert artifacts to a string or dictionary.

Prerequisites

Before you begin,

- Connect to H2O MLOps. For instructions, see Connect to H2O MLOps.
- Create an MLOps entity and assign it to a variable named `entity`.
  - To create a workspace, see Create a workspace.
  - To create an experiment entity, see Create an experiment.

Add an artifact

Use the `add()` method to upload an artifact to an entity.

```
artifact = entity.artifacts.add(
   data="/path/to/docx_artifact.docx",
   mime_type="application/vnd.openxmlformats-officedocument.wordprocessingml.document",
)
```

Note: If you link a H2O Driverless AI (DAI) experiment to a workspace directly from a DAI instance, all its artifacts are added to the experiment automatically.

View artifacts

List artifacts

Use the `list()` method to view all artifacts linked to an entity.

Input:

```
artifacts = entity.artifacts.list()
artifacts
```

Output:

```
   | name               | mime_type                  | uid
---+--------------------+----------------------------+--------------------------------------
 0 | docx_artifact.docx | application/vnd.openxmlfo | 92e4ea49-fdcd-436b-9293-dcd0e8fdae18
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(artifacts)` returns the total number of rows it contains, not just the number currently displayed.
- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: `>python >artifacts.show(n=100) >` This will display up to 100 artifacts.
- The `artifacts` can be iterated over, as it is designed to behave like an iterator.

Filter artifacts

Use the `list()` method with key-value arguments to filter the artifacts.

Input:

```
entity.artifacts.list(name="docx_artifact.docx")
```

This returns a list of matching artifacts as a table.

Output:

```
   | name               | mime_type                  | uid
---+--------------------+----------------------------+--------------------------------------
 0 | docx_artifact.docx | application/vnd.openxmlfo | 92e4ea49-fdcd-436b-9293-dcd0e8fdae18
```

Retrieve an artifact

Use the `get()` method to retrieve a specific artifact by its unique ID.

Input:

```
artifact = entity.artifacts.get(uid="92e4ea49-fdcd-436b-9293-dcd0e8fdae18")
artifact
```

Note: You can retrieve a specific artifact from the list returned by `list()` using indexing.
For example, `artifact = entity.artifacts.list(key=value)[index]`. The `key` and `value` arguments are optional.

Output:

```
<class 'h2o_mlops._artifacts.MLOpsArtifact(
   uid='92e4ea49-fdcd-436b-9293-dcd0e8fdae18',
   parent_entity_uid='777b3d15-36ba-4a96-b687-47e3d6687d4d',
   name='docx_artifact.docx',
   state='AVAILABLE',
   mime_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document',
   size=0,
   md5_digest='1B2M2Y8AsgTpgAmY7PhCfg==',
   created_time=datetime.datetime(2025, 6, 11, 17, 31, 12, 850635, tzinfo=tzutc()),
   uploaded_time=datetime.datetime(2025, 6, 11, 17, 31, 16, 26743, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 11, 17, 31, 16, 27248, tzinfo=tzutc()),
)'>
```

Artifact properties

An artifact has the following main properties:

- `uid`: Unique ID of the artifact
- `parent_entity_uid`: ID of the parent workspace, dataset, etc.
- `name`: Name of the artifact
- `state`: Artifact state
- `mime_type`: File type of the artifact
- `size`: File size in bytes
- `md5_digest`: MD5 hash of the artifact
- `created_time`: Timestamp when the artifact was created
- `uploaded_time`: Timestamp when the artifact was uploaded
- `last_modified_time`: Last time the artifact was updated
- `model_info`: Model-specific metadata, if applicable

Download an artifact

To download an artifact:

Input:

```
artifact.download()
```

Output:

```
'docx_artifact.docx'
```

The `download()` method supports the following optional parameters:

- `directory`: Target folder (default: current working directory)
- `file_name`: Output filename (default: artifact's name)
- `overwrite`: Whether to overwrite the file if it exists (default: `False`)
- `buffer`: Store the file in memory using `io.BytesIO` instead of writing to disk

Note:

- If you don't specify any options, the artifact downloads to the current working directory using its name.
- If the artifact name does not include a file extension, you must pass the `file_name` argument with the appropriate extension. You can use the artifact's MIME type (`mime_type`) to determine the correct file extension.

  Common MIME types and corresponding file extensions:

MIME type	File extension
application/vnd.openxmlformats-officedocument.wordprocessingml.document	`.docx`
application/json	`.json`
text/plain	`.txt`
application/zip	`.zip`
application/pdf	`.pdf`
image/png	`.png`

Convert artifacts

Artifacts can be converted to string or dictionary format, depending on their type:

- A JSON artifact can be converted to a string or a dictionary.
- A text artifact can **only** be converted to a string.
- Other artifact types cannot be converted.

Convert JSON artifacts

To convert JSON artifacts to a string and a dictionary, follow these steps:

1. Add the JSON file to the specified entity as an artifact.

   ```
   json_artifact = entity.artifacts.add(
      data="/path/to/json_artifact.json",
      mime_type="application/json",
   )
   ```

2. Convert the JSON artifact to a string:

   Input:

   ```
   json_artifact.to_string()
   ```

   Output:

   ```
   '{\n    "key": "value"\n}\n'
   ```

3. Convert a JSON artifact to a dictionary:

   Input:

   ```
   json_artifact.to_dict()
   ```

   Output:

   ```
   {'key': 'value'}
   ```

Convert text artifacts

To convert text artifacts to a string, follow these steps:

1. Add the text file to the specified entity as an artifact.

```
text_artifact = entity.artifacts.add(
   data="/path/to/txt_artifact.txt",
   mime_type="text/plain",
)
```

2. Convert the text artifact to a string:

   Input:

```
text_artifact.to_string()
```

   Output:

```
"It's a text file.\n"
```

Update an artifact

You can update only the `name` and `parent_entity` fields of an artifact.

Make sure to retrieve the artifact before updating it. See Retrieve an artifact.

Input:

```
artifact.update(name="my-docx-artifact")
artifact
```

Output:

```
<class 'h2o_mlops._artifacts.MLOpsArtifact(
   uid='92e4ea49-fdcd-436b-9293-dcd0e8fdae18',
   parent_entity_uid='777b3d15-36ba-4a96-b687-47e3d6687d4d',
   name='my-docx-artifact',
   state='AVAILABLE',
   mime_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document',
   size=0,
   md5_digest='1B2M2Y8AsgTpgAmY7PhCfg==',
   created_time=datetime.datetime(2025, 6, 11, 17, 31, 12, 850635, tzinfo=tzutc()),
   uploaded_time=datetime.datetime(2025, 6, 11, 17, 31, 16, 26743, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 11, 17, 31, 36, 773648, tzinfo=tzutc()),
)'>
```

Delete an artifact

This section describes how to delete an artifact.

Use the `delete()` method to remove the artifact from the entity:

Input:

```
artifact.delete()
entity.artifacts.list()
```

In this example, `artifact.delete()` deletes the `docx_artifact.docx` artifact assigned to the artifact variable earlier in the Add an artifact section and retrieved in the Retrieve an artifact section.

The `list()` output confirms that the artifact has been successfully removed.

Output:

```
   | name               | mime_type        | uid
---+--------------------+------------------+--------------------------------------
 0 | txt_artifact.txt   | text/plain       | 04eaf9e2-f6cf-4140-9686-b31290f7fd17
 1 | json_artifact.json | application/json | 5d442fc2-d0ae-42eb-8050-42255fa8c10a
```

After deletion, the `docx_artifact.docx` artifact no longer appears in the list of artifacts for the entity.

Manage Models

This page describes how to create and view models, manage model versions, and delete models using the H2O MLOps Python client.

To learn more about models, see Understand models.

Prerequisites

Before you begin, 1. Connect to H2O MLOps. For instructions, see Connect to H2O MLOps. 2. Create a workspace. For instructions, see Create a workspace. 3. Create an experiment. For instructions, see Create an experiment.

Create a model

Create a model within the workspace using the `create()` method by specifying the model name and the description.

```
model = workspace.models.create(name="my-model", description="My Model")
```

Note: The name of the model must be unique within the workspace.

View models

Count models

Get the total number of models in a workspace:

Input:

```
workspace.models.count()
```

Output:

```
1
```

List models

List all models in a workspace:

Input:

```
models = workspace.models.list()
models
```

Output:

```
   | name     | uid
---+----------+-------------------------------------
 0 | my-model | 27965199-6f0f-4cbe-bf48-e59b05fa1f04
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(models)` returns the total number of rows it contains, not just the number currently displayed.
- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: >`python` `>models.show(n=100)` `>` This will display up to 100 models.
- The `models` can be iterated over, as it is designed to behave like an iterator.

Filter models

Use the `list()` method with key-value arguments to filter the models.

Input:

```
workspace.models.list(name="my-model")
```

This returns a list of matching models as a table.

Output:

```
   | name     | uid
---+----------+--------------------------------------
 0 | my-model | 27965199-6f0f-4cbe-bf48-e59b05fa1f04
```

Retrieve a model

Retrieve a model by UID:

```
model = workspace.models.get(uid="d9a47c99-c66c-4ff9-b2b6-30faf5f413ef")
```

Retrieve a model by model name:

Input:

```
model = workspace.models.get(name="my-model")
model
```

Output:

```
<class 'h2o_mlops._models.MLOpsModel(
   uid='27965199-6f0f-4cbe-bf48-e59b05fa1f04',
   name='my-model',
   description='My Model',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 6, 3, 15, 33, 10, 882595, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 3, 15, 33, 10, 882595, tzinfo=tzutc()),
)'>
```

Note: You can also retrieve a model from the list returned by `list()` using indexing.
For example, `model = workspace.models.list(key=value)[index]`. The `key` and `value` arguments are optional.

Model properties

A model has the following main properties:

- `uid`: The unique identifier for the model.
- `name`: The name of the model.
- `description`: A description of the model.
- `creator`: The user who created the model.
- `created_time`: The timestamp when the model was created.
- `last_modified_time`: The timestamp of the last modification.
- `last_modified_by`: The user who last modified the model.
- `version_count`: The number of versions of the model.

Update a model

You can update only the `name` and `description` fields of a model.

Make sure to retrieve the model instance before updating it. See Retrieve a model.

Input:

```
model.update(name="new-my-model", description="New My Model")
model
```

Output:

```
<class 'h2o_mlops._models.MLOpsModel(
   uid='27965199-6f0f-4cbe-bf48-e59b05fa1f04',
   name='new-my-model',
   description='New My Model',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 6, 3, 15, 33, 10, 882595, tzinfo=tzutc()),
```

```
    last_modified_time=datetime.datetime(2025, 6, 3, 15, 33, 32, 502773, tzinfo=tzutc()),
)'>
```

Manage model versions

Register an experiment with a model

To create a new version of a model, register an experiment with it.

Use the following code:

```
model.register(experiment=experiment)
```

Note: You can also register an experiment using the following alternative methods:

- Register an experiment with an existing model using artifact data. No need for an `MLOpsExperiment` instance. This method creates and registers the experiment with the model.

  Example: `>python >model.register(experiment="/path/experiment.zip", name="experiment-name") >`

- Register an experiment and create a new model directly using artifact data. No need for `MLOpsModel` or `MLOpsExperiment` instances. This method creates both the model and the experiment, then registers the experiment with the model.

  Example: `>python >model = workspace.models.register(experiment="/path/experiment.zip", name="experiment-and-model-name",) >`

To verify that the experiment is registered with the model, run the following code:

Input:

```
experiment.registered_model
```

This returns a tuple containing the associated `MLOpsModel` instance and the model version the experiment was registered to.

Output:

```
(<class 'h2o_mlops._models.MLOpsModel(
    uid='27965199-6f0f-4cbe-bf48-e59b05fa1f04',
    name='new-my-model',
    description='New My Model',
    creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
    created_time=datetime.datetime(2025, 6, 3, 15, 33, 10, 882595, tzinfo=tzutc()),
    last_modified_time=datetime.datetime(2025, 6, 3, 15, 33, 40, 578744, tzinfo=tzutc()),
)'>,1)
```

List model versions

List all versions of a model:

Input:

```
model.versions()
```

Output:

```
   |   version | experiment_uid
---+-----------+--------------------------------------
 0 |         1 | bdf42b57-c3de-4f33-8c37-f19a9f5f765d
```

Filter model versions

Use the `versions()` method with key-value arguments to filter the model versions.

Input:

```
model.versions(version=1)
```

This returns a list of matching model versions as a table.

Output:

```
   |   version | experiment_uid
---+-----------+--------------------------------------
 0 |         1 | bdf42b57-c3de-4f33-8c37-f19a9f5f765d
```

Retrieve a model version

You can retrieve a specific version of a model from the list returned by `versions()` using indexing.
For example, `version = model.versions(key=value)[index]`. The `key` and `value` arguments are optional.

Input:

```
model.versions(version=1)[0]
```

Output:

```
MLOpsModelVersion(
   uid='7fb119cf-aaaa-42e2-985c-e71a84e6396a',
   version=1,
   model_uid='27965199-6f0f-4cbe-bf48-e59b05fa1f04'
   experiment_uid='bdf42b57-c3de-4f33-8c37-f19a9f5f765d',
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time='2025-06-03 03:33:40 PM',
   last_modified_time='2025-06-03 03:33:40 PM',
   last_modified_by='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   state='ACTIVE'
)
```

Retrieve the experiment

Get the experiment associated with a model version:

Input:

```
model.experiment(model_version=1)
```

Note: If nothing is specified for the `model_version`, the default value `"latest"` is used.

Output:

```
<class 'h2o_mlops._experiments.MLOpsExperiment(
   uid='bdf42b57-c3de-4f33-8c37-f19a9f5f765d',
   name='H2O3 MOJO experiment',
   description='GLM - Regression',
   is_registered=True,
   owner_username='test.user@test.com',
   created_time=datetime.datetime(2025, 6, 3, 15, 33, 8, 159914, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 3, 15, 33, 8, 159914, tzinfo=tzutc())
)'>
```

Unregister an experiment from a model

To remove an experiment from a model, run:

```
model.unregister(experiment=experiment)
```

After unregistering the experiment, list the available model versions to verify the change:

Input:

```
model.versions()
```

Output:

```
   | version   | experiment_uid
---+-----------+------------------
```

The model version created during registration is removed.

Note: To unregister all the models, use: >`python >model.unregister(unregister_all=True) >`

Delete models

This section describes two ways to delete models.

danger WARNING Deleting a model also deletes all of its associated versions. This action is irreversible.

Delete using a model instance

If you already have a reference to the `model` object, use the `delete()` method:

```
model.delete()
```

Delete using model UIDs

You can delete multiple models at once by specifying their UIDs.

Input:

```
workspace.models.delete(uids=["c62ced74-905a-4f06-856d-33b9f5725901"])
```

Note: You can also pass a list of `MLOpsModel` instances or a `_utils.Table` containing models. For example: `workspace.models.delete(models=[model])`.

Output:

```
   | model_uid                            | is_deleted   | message
---+--------------------------------------+--------------+-----------
 0 | c62ced74-905a-4f06-856d-33b9f5725901 | True         |
```

Configure deployments

This page describes the available options for configuring deployments using the H2O MLOps Python client.

Prerequisites

Before you begin, complete the following steps:

1. Import the necessary Python packages. For instructions, see Step 1: Import the required packages.
2. Connect to H2O MLOps. For instructions, see Connect to H2O MLOps.
3. Create a workspace. For instructions, see Create a workspace.
4. Create one or two experiments. For instructions, see Create an experiment.
5. Create models and register the experiments with them. For instructions, see Register an experiment with a model.

To follow the examples in the next sections, assume the following:

- You have created two models and assigned them to the variables `model_1` and `model_2`. Each model has a registered experiment.
- You have two scoring runtimes assigned to the variables `scoring_runtime_1` and `scoring_runtime_2`, each corresponding to the respective experiment.

For details on how to retrieve a scoring runtime for an experiment, see Scoring runtimes.

Composition options

Composition options define how models are composed for deployment.

- `model`: `_models.MLOpsModel` - The model to deploy.
- `scoring_runtime`: `_runtimes.MLOpsScoringRuntime` - The runtime environment used for scoring.
- `model_version`: `Union[int, str] = "latest"` - The version of the model to deploy.
- `traffic_weight`: `Optional[int] = None` - The ratio of traffic to direct to a specific deployment in an A/B test.
- `primary`: `Optional[bool] = None` - Indicates whether this deployment is the primary (champion) or secondary (challenger) in a champion/challenger setup.

There are three types of deployments: **single model**, **A/B test**, and **champion/challenger**. Each type requires different composition options, as described below:

Single model deployment

```
composition_options = options.CompositionOptions(
   model=model_1,
   scoring_runtime=scoring_runtime_1,
)
```

A/B test deployment

```
composition_options = [
   options.CompositionOptions(
       model=model_1,
       scoring_runtime=scoring_runtime_1,
       traffic_weight=2,
   ),
   options.CompositionOptions(
       model=model_2,
       scoring_runtime=scoring_runtime_2,
       traffic_weight=1,
   ),
]
```

Champion/Challenger deployment

```
composition_options = [
   options.CompositionOptions(
       model=model_1,
       scoring_runtime=scoring_runtime_1,
```

```
            primary=True,
        ),
        options.CompositionOptions(
            model=model_2,
            scoring_runtime=scoring_runtime_2,
            primary=False,
        ),
    ]
```

For more information about deployment types, see Deployment type.

Security options

Security options let you configure security settings for your deployment.

- `security_type`: `types.SecurityType` - The type of security to use.
  Available values:
  - `DISABLED`
  - `PLAIN_PASSPHRASE`
  - `HASHED_PASSPHRASE`
  - `OIDC_AUTH`: Requires additional configuration in the `values.yaml` file.
- `passphrase`: `Optional[str] = None` - The passphrase to use, if required by the selected security type.

For more information, see Endpoint security.

Note: Not all types are supported in every environment. Support can be configurable.

To check the allowed types using the H2O MLOps Python client, run: `mlops.configs.allowed_security_types`

Use the following code to create security options:

```
security_options = options.SecurityOptions(
   security_type=types.SecurityType.HASHED_PASSPHRASE,
   passphrase="123abcABC",
)
```

Kubernetes options

The following options let you customize Kubernetes deployment settings:

- `replicas`: `int = 1`
- `requests`: `Optional[Dict[str, str]] = None`
- `limits`: `Optional[Dict[str, str]] = None`
- `affinity`: `Optional[str] = None`
- `toleration`: `Optional[str] = None`

For more information about `replicas`, `requests`, and `limits`, see Kubernetes options. For more information about `affinity` and `toleration`, see Node affinity and toleration.

Note: Not all options are supported in every environment. Support can be configurable.

To check the default / allowed values for each option using the H2O MLOps Python client, run the following codes:

- `mlops.configs.default_k8s_requests`
- `mlops.configs.default_k8s_limits`
- `mlops.configs.allowed_k8s_affinities`
- `mlops.configs.allowed_k8s_tolerations`

Use the following code to create Kubernetes options:

```
kubernetes_options = options.KubernetesOptions()
```

Vertical Pod Autoscaler (VPA) options

Configure the following Vertical Pod Autoscaler (VPA) settings to automatically adjust resource requests:

- `resource_type`: `types.KubernetesResourceType` - The type of resource to adjust.
  Supported values are:
  - `CPU`
  - `MEMORY`
- `unit`: `types.KubernetesResourceUnitType` - The unit used for the resource.
  Supported values are:
  - `MILLI_CORE`
  - `CORES`
  - `MIB`
  - `GIB`
- `min_bound`: `float` - The minimum resource request value allowed.
- `max_bound`: `float` - The maximum resource request value allowed.

Use the following code to create VPA options:

```
vpa_options = [
   options.VPAOptions(
       resource_type=types.KubernetesResourceType.CPU,
       unit=types.KubernetesResourceUnitType.MILLI_CORES,
       min_bound=100,
       max_bound=200,
   ),
   options.VPAOptions(
       resource_type=types.KubernetesResourceType.MEMORY,
       unit=types.KubernetesResourceUnitType.MIB,
       min_bound=200,
       max_bound=400,
   ),
]
```

Pod Disruption Budget (PDB) options

Use the following options to control pod availability during voluntary disruptions:

- `pods`: `int` - The number of pods.
- `disruption_policy`: `types.DisruptionPolicyType` - The disruption policy to apply.
  Valid values:
  - `MIN_AVAILABLE`
  - `MAX_UNAVAILABLE`
- `is_percentage`: `bool = False` - Indicates whether the pods value is a percentage.

For more information, see Pod Disruption Budget (PDB).

Use the following code to create PDB options:

```
pdb_options = options.PDBOptions(
   pods=1,
   disruption_policy=types.DisruptionPolicyType.MIN_AVAILABLE,
)
```

Environment variables

Specify environment variables to add to the scoring runtime:

```
environment_variables = {
   "KEY_1": "VALUE_1",
   "KEY_2": "VALUE_2",
   "KEY_3": "VALUE_3",
}
```

CORS origins

Define allowed CORS origins:

```
cors_origins = [
   "http://localhost:8080",
   "http://customcors.com",
]
```

Monitoring options

Configure monitoring settings for your deployment:

```
monitoring_options = options.MonitoringOptions()
```

Manage deployments

This page describes how to create, view, update, and delete deployments, as well as how to view logs and configure endpoints using the H2O MLOps Python client.

To learn more about deployments, see Understand deployments in MLOps.

Prerequisites

Before you begin,

1. Import the necessary Python packages. For instructions, see Step 1: Import the required packages.
2. Connect to H2O MLOps. For instructions, see Connect to H2O MLOps.
3. Create a workspace. For instructions, see Create a workspace.
4. Create one or two experiments. For instructions, see Create an experiment.
5. Create models and register the experiments with them. For instructions, see Register an experiment with a model.

Create a deployment

Create a deployment within the workspace using the `create()` method by specifying the deployment name, composition options, and security options.

```
deployment = workspace.deployments.create(
   name="my-deployment",
   composition_options=options.CompositionOptions(
       model=model,
       scoring_runtime=mlops.runtimes.scoring.get(
           artifact_type="h2o3_mojo",
           runtime_uid="h2o3_mojo_runtime",
       ),
   ),
   security_options=options.SecurityOptions(
       security_type=types.SecurityType.DISABLED,
   ),
)
```

The `create()` method supports the following optional arguments:

- `mode: types.DeploymentModeType = types.DeploymentModeType.SINGLE_MODEL` – The type of deployment. Available values:
  - `SINGLE_MODEL`
  - `AB_TEST`
  - `CHAMPION_CHALLENGER`
- `description: Optional[str] = None` – The deployment description.
- `kubernetes_options: Optional[options.KubernetesOptions] = None` – Customize Kubernetes deployment settings.
- `vpa_options: Optional[List[options.VPAOptions]] = None` – Configure Vertical Pod Autoscaler (VPA) settings.
- `pdb_options: Optional[options.PDBOptions] = None` – Control pod availability during voluntary disruptions.
- `environment_variables: Optional[Dict[str, str]] = None` – Specify environment variables to add to the scoring runtime.
- `cors_origins: Optional[List[str]] = None` – Define allowed CORS origins.
- `monitoring_options: Optional[options.MonitoringOptions] = None` – Configure monitoring settings for the deployment.

For more details on these configuration options, see Configure deployments.

The deployment might take a few seconds. Use the following command to wait until the deployment is healthy and ready to use:

```
deployment.wait_for_healthy()
```

Note: To retry a failed deployment, use the following command. It checks the current deployment status and, if it's in a failed state, attempts to redeploy it: >`python >deployment.redeploy_if_failed() >`

View deployments

List deployments

List all deployments in a workspace:

Input:

```
deployments = workspace.deployments.list()
deployments
```

Output:

```
   | name          | mode         | uid
---+---------------+--------------+--------------------------------------
 0 | my-deployment | Single Model | 48ece40f-8608-473a-92a6-388e164e995
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(deployments)` returns the total number of rows it contains, not just the number currently displayed.
- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: >`python >deployments.show(n=100) >` This will display up to 100 deployments.
- The `deployments` can be iterated over, as it is designed to behave like an iterator.

List deployment statuses

Use the `statuses()` method to view the statuses of all deployments.

Input:

```
workspace.deployments.statuses()
```

This returns a list of deployments and their statuses in a table.

Output:

```
   | uid                                  | state
---+--------------------------------------+---------
 0 | 48ece40f-8608-473a-92a6-388e164e9952 | HEALTHY
```

Possible status values:

- `PREPARING`: Preparing the deployment for launch (for example, building assets).
- `LAUNCHING`: Deployment is launching to an environment (for example, waiting for the environment to start).
- `FAILED`: Deployment failed during preparation or launch.
- `HEALTHY`: Deployment is alive and healthy.
- `UNHEALTHY`: Health issues detected in the launched deployment.
- `TERMINATING`: Deployment is terminating (that is, environment resources are being brought down).
- `PENDING`: Deployment created and awaiting processing.
- `STOPPED`: The deployment is scaled down.

Filter deployments

Use the `list()` method with key-value arguments to filter deployments.

Input:

```
workspace.deployments.list(name="my-deployment")
```

This returns a list of matching deployments as a table.

Output:

```
   | name          | mode         | uid
---+---------------+--------------+--------------------------------------
 0 | my-deployment | Single Model | 48ece40f-8608-473a-92a6-388e164e9952
```

Retrieve a deployment

Retrieve a deployment by UID:

Input:

```
deployment = workspace.deployments.get(uid="48ece40f-8608-473a-92a6-388e164e9952")
deployment
```

Note: You can also retrieve a deployment from the list returned by `list()` using indexing.
For example, `deployment = workspace.deployments.list(key=value)[index]`. The `key` and `value` arguments are optional.

Output:

```
<class 'h2o_mlops._deployments.MLOpsScoringDeployment(
   uid='48ece40f-8608-473a-92a6-388e164e9952',
   name='my-deployment',
   description='',
   mode=<DeploymentModeType.SINGLE_MODEL: 'Single Model'>,
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 6, 27, 5, 4, 32, 14134, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 27, 5, 4, 32, 14134, tzinfo=tzutc()),
)'>
```

Deployment properties

A deployment has the following main properties:

- `uid`: The unique identifier for the deployment.
- `name`: The name of the deployment.
- `description`: A description of the deployment.
- `mode`: The deployment mode (for example, `SINGLE_MODEL` or `AB_TEST`).
- `creator`: The user who created the deployment.
- `created_time`: The timestamp when the deployment was created.
- `last_modified_time`: The timestamp of the last update.
- `revision_uid`: The revision ID of the deployment.
- `state`: The current status of the deployment.
- `is_healthy`: A boolean indicating whether the deployment is healthy.
- `composition_options`: Define how models are composed for deployment.
- `security_options`: Security configuration options.
- `kubernetes_options`: Kubernetes deployment configuration.
- `vpa_options`: Vertical Pod Autoscaler (VPA) settings.
- `pdb_options`: Pod disruption budget settings.
- `environment_variables`: Environment variables to add to the scoring runtime.
- `cors_origins`: A list of allowed CORS origins.
- `monitoring_options`: Monitoring configuration.
- `experiments`: The associated experiment(s) for the deployment.

View deployment logs

Call the `logs()` method to access the logs for a deployment:

```
logs = deployment.logs()
```

This returns a dictionary of log entries from different deployment pods.

To see which pods' log entries were retrieved, use the `keys()` method:

Input:

```
keys = list(logs.keys())
keys
```

Output:

```
['primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.artifact-fetcher',
'primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.artifact-processor',
'primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.events',
'primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.proxy',
'primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.runtime']
```

As you already know the available dictionary keys, you can access the log entries for a specific pod.

For example, to view the full log for the `runtime` pod:

Input:

```
logs["primary.pr-48ece40f-8608-473a-92a6-388e164e9952-769f88644b-7qzcs.runtime"]
```

Output:

```
['2025-06-27 05:04:36.423321198 +0000 UTC: Picked up _JAVA_OPTIONS: -Dmojo.path=/data/model',
'2025-06-27 05:04:36.63684369 +0000 UTC: Standard Commons Logging discovery in action with spring-jcl: p
'2025-06-27 05:04:37.092592485 +0000 UTC: ',
'2025-06-27 05:04:37.092627005 +0000 UTC:   .   ____          _            __ _ _',
"2025-06-27 05:04:37.092634066 +0000 UTC:  /\\\\ / ___'_ __ _ _(_)_ __  __ _ \\ \\ \\ \\",
"2025-06-27 05:04:37.092639646 +0000 UTC: ( ( )\\___ | '_ | '_| | '_ \\/ _` | \\ \\ \\ \\",
'2025-06-27 05:04:37.092645106 +0000 UTC:  \\\\/  ___)| |_)| | | | | || (_| |  ) ) ) )',
"2025-06-27 05:04:37.092650616 +0000 UTC:   '  |____| .__|_| |_|_| |_\\__, | / / / /",
'2025-06-27 05:04:37.092656126 +0000 UTC:  =========|_|==============|___/=/_/_/_/',
'2025-06-27 05:04:37.092661606 +0000 UTC: ',
'2025-06-27 05:04:37.094651073 +0000 UTC:  :: Spring Boot ::                (v3.3.12)',
'2025-06-27 05:04:37.094674983 +0000 UTC: ',
'2025-06-27 05:04:37.143507059 +0000 UTC: 2025-06-27T05:04:37.142Z  INFO 1 --- [           main] a.h.m.d
'2025-06-27 05:04:37.144078602 +0000 UTC: 2025-06-27T05:04:37.143Z  INFO 1 --- [           main] a.h.m.d
'2025-06-27 05:04:37.974359799 +0000 UTC: 2025-06-27T05:04:37.974Z  INFO 1 --- [           main] o.s.b.w
'2025-06-27 05:04:37.984421453 +0000 UTC: 2025-06-27T05:04:37.984Z  INFO 1 --- [           main] o.apach
'2025-06-27 05:04:37.984547596 +0000 UTC: 2025-06-27T05:04:37.984Z  INFO 1 --- [           main] o.apach
'2025-06-27 05:04:38.01826987 +0000 UTC: 2025-06-27T05:04:38.017Z  INFO 1 --- [           main] o.a.c.c.
'2025-06-27 05:04:38.01910209 +0000 UTC: 2025-06-27T05:04:38.018Z  INFO 1 --- [           main] w.s.c.Se
'2025-06-27 05:04:38.021103686 +0000 UTC: Standard Commons Logging discovery in action with spring-jcl:
'2025-06-27 05:04:38.071485518 +0000 UTC: 2025-06-27T05:04:38.070Z  INFO 1 --- [           main] a.h.m.d
'2025-06-27 05:04:38.075504291 +0000 UTC: 2025-06-27T05:04:38.075Z  INFO 1 --- [           main] a.h.m.r
'2025-06-27 05:04:38.165548915 +0000 UTC: 2025-06-27T05:04:38.165Z  INFO 1 --- [           main] a.h.m.d
'2025-06-27 05:04:38.165589516 +0000 UTC: 2025-06-27T05:04:38.165Z  INFO 1 --- [           main] a.h.m.r
'2025-06-27 05:04:38.18428143 +0000 UTC: 2025-06-27T05:04:38.183Z  INFO 1 --- [           main] a.h.m.d.
"2025-06-27 05:04:38.486280193 +0000 UTC: 2025-06-27T05:04:38.485Z  INFO 1 --- [           main] o.s.b.w
'2025-06-27 05:04:38.50331213 +0000 UTC: 2025-06-27T05:04:38.502Z  INFO 1 --- [           main] a.h.m.d.
"2025-06-27 05:04:39.403663076 +0000 UTC: 2025-06-27T05:04:39.403Z  INFO 1 --- [nio-8080-exec-1] o.a.c.c
"2025-06-27 05:04:39.4038616 +0000 UTC: 2025-06-27T05:04:39.403Z  INFO 1 --- [nio-8080-exec-1] o.s.web.s
'2025-06-27 05:04:39.40514298 +0000 UTC: 2025-06-27T05:04:39.404Z  INFO 1 --- [nio-8080-exec-1] o.s.web.
```

View deployment logs from a specific time

To get logs starting from a specific date and time:

```
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo
```

```
since_time = datetime(2025, 6, 25, 9, 0, 0, tzinfo=ZoneInfo("Asia/Colombo"))
logs = deployment.logs(since_time=since_time)
```

Note: You can retrieve logs relative to the deployment's creation time by using `timedelta`. For example, to get logs starting 5 minutes after the deployment was created: `>python >since_time = deployment.created_time + timedelta(minutes=5) >logs = deployment.logs(since_time=since_time) >`

Alternatively, you can get recent logs by subtracting a time duration from the current time. Make sure to use timezone-aware datetime values. For example, to get logs from the past 5 minutes using the `Asia/Colombo` timezone: `>python >since_time = datetime.now(ZoneInfo("Asia/Colombo")) - timedelta(minutes=5) >logs = deployment.logs(since_time=since_time) >`

Manage endpoint

Configure endpoint

Configure a static path for the MLOps deployment REST endpoint:

```
endpoint = deployment.configure_endpoint(
   path="static-path",
)
```

Note:

- You can use the configured path as an alias for the deployment ID in the deployment scorer API base URL.
- This path must be globally unique within the H2O MLOps environment to ensure proper routing and avoid collisions.
- For example, while the default scoring endpoint, `f"https://model.dummy-env.h2o.ai/{deployment.uid}/model/score"` is always available and pre-configured, you can also use `f"https://model.dummy-env.h2o.ai/{endpoint.path}/model/score"` for scoring.
- Both endpoints are valid and functional. The same applies to other endpoint types.

The `configure_endpoint()` method supports the following arguments:

- `path`: `str` - Path to use for the target deployment URLs.
- `name`: `Optional[str] = None` - Display name for the MLOps endpoint. (Used only if a new endpoint is created.)
- `description`: `Optional[str] = None` - Description for the MLOps endpoint. (Used only if a new endpoint is created.)
- `force`: `bool = False` - Whether to attempt to reassign the path if it's already in use by another deployment.

List deployment endpoints

To list all the endpoints attached to the deployment:

Input:

```
endpoints = deployment.endpoints
endpoints
```

Output:

```
   | name        | path        | uid                                  | target_deployment_uid
---+-------------+-------------+--------------------------------------+-----------------------------
 0 | static-path | static-path | 6ab2b92e-98c0-4c38-b4ef-4f6d9ba66e6f | 48ece40f-8608-473a-92a6-388e164e
```

Note: To list all the endpoints available within the workspace:

```
endpoints = workspace.endpoints.list()
```

Note:

- The output of `list()` method is displayed in a neatly formatted view. By default, only the first 50 rows are displayed to keep the output concise and manageable.
- Calling `len(endpoints)` returns the total number of rows it contains, not just the number currently displayed.
- To customize the number of rows displayed, you can call the `show()` method with the `n` argument. This allows more rows to be shown when needed. For example: `>python >endpoints.show(n=100) >` This will display up to 100 endpoints.

- The `endpoints` can be iterated over, as it is designed to behave like an iterator.

Retrieve a deployment endpoint

To retrieve a deployment endpoint:

Input:

```
endpoint = endpoints[0]
endpoint
```

Output:

```
<class 'h2o_mlops._endpoints.MLOpsEndpoint(
   uid='6ab2b92e-98c0-4c38-b4ef-4f6d9ba66e6f',
   name='static-path',
   description='',
   path='static-path',
   created_time=datetime.datetime(2025, 6, 27, 5, 5, 40, 146102, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 27, 5, 5, 40, 146481, tzinfo=tzutc()),
)'>
```

Note: To retrieve an endpoint available within the workspace: >`python >workspace.endpoints.get(uid=...) >`

Detach a configured endpoint

To detach an endpoint from its current deployment, unset its target deployment as shown below:

```
endpoint.update(target_deployment=None)
```

You can then reuse the detached endpoint with another deployment without recreating it.

To verify that the endpoint is detached:

Input:

```
deployment.endpoints
```

Output:

```
   | name   | path   | uid   | target_deployment_uid
---+--------+--------+-------+-------------------------
```

Delete an endpoint

danger warning Deleting an endpoint also detaches the deployment associated with it.

To delete the endpoint:

```
endpoint.delete()
```

To verify deletion:

Input:

```
workspace.endpoints.list()
```

Output:

```
   | name   | path   | uid   | target_deployment_uid
---+--------+--------+-------+-------------------------
```

Update a deployment

You can update a deployment's properties, including `name`, `description`, `security_options`, `kubernetes_options`, `vpa_options`, `pdb_options`, `environment_variables`, `cors_origins`, and `monitoring_options`. For details on how to configure these properties, see Configure deployments.

Make sure to retrieve the deployment before updating it. See Retrieve a deployment.

Input:

```
deployment.update(name="my-new-deployment")
deployment
```

Output:

```
<class 'h2o_mlops._deployments.MLOpsScoringDeployment(
   uid='48ece40f-8608-473a-92a6-388e164e9952',
   name='my-new-deployment',
   description='',
   mode=<DeploymentModeType.SINGLE_MODEL: 'Single Model'>,
   creator_uid='4c4eb198-bcbc-4442-91f6-a27deb53e9c1',
   created_time=datetime.datetime(2025, 6, 27, 5, 4, 32, 14134, tzinfo=tzutc()),
   last_modified_time=datetime.datetime(2025, 6, 27, 5, 6, 7, 730844, tzinfo=tzutc()),
)'>
```

Delete a deployment

Note: Deleting a deployment doesn't delete the endpoint attached to it. Instead, the endpoint becomes detached and enters a dangling state. You can reuse the same endpoint in a future deployment.

To delete a deployment:

```
deployment.delete()
```

Deployment scorer

You can use the H2O MLOps Python client to score against deployments. This page explains how to retrieve deployment scorer, access the scorer's endpoints, and send scoring requests using various capabilities such as prediction intervals, Shapley values, and media inputs.

Prerequisites

Before you begin,

1. Import the necessary Python packages. For instructions, see Step 1: Import the required packages.
2. Connect to H2O MLOps. For instructions, see Connect to H2O MLOps.
3. Create a workspace. For instructions, see Create a workspace.
4. Create one or two experiments. For instructions, see Create an experiment.
5. Create models and register the experiments with them. For instructions, see Register an experiment with a model.
6. Create a deployment. For instructions, see Create a deployment.

View deployment scorers

List deployment scorers

List all deployment scorers in a workspace:

Input:

```
workspace.deployments.scorers()
```

Output:

```
   | uid                                  | scoring_endpoint
---+--------------------------------------+-----------------------------------------------------------------
 0 | 822fcf3d-6d4c-4948-a71b-2d0b46db82a9 | https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-
```

Filter deployment scorers

Use the `scorers()` method with key-value arguments to filter the deployment scorers.

Input:

```
workspace.deployments.scorers(uid="822fcf3d-6d4c-4948-a71b-2d0b46db82a9")
```

This returns a list of matching deployment scorers as a table.

Output:

```
   | uid                                  | scoring_endpoint
---+--------------------------------------+-----------------------------------------------------------------
 0 | 822fcf3d-6d4c-4948-a71b-2d0b46db82a9 | https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-
```

Retrieve a deployment scorer

To retrieve the scorer object for a deployment:

Input:

```
scorer = deployment.scorer
scorer
```

Note: You can also retrieve a deployment scorer from the list returned by `scorers()` using indexing.
For example, `scorer = workspace.deployments.scorers(key=value)[index]`. The `key` and `value` arguments are optional.

Output:

```
<class 'h2o_mlops._deployments.MLOpsDeploymentScorer(
   api_base_url='https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-a71b-2d0b46db82a9',
   capabilities_endpoint='https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-a71b-2d0b46db82a9/
   schema_endpoint='https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-a71b-2d0b46db82a9/model/
```

```
    sample_request_endpoint='https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-a71b-2d0b46db82a
    scoring_endpoint='https://model.dev.mlops-internal.h2o.dev/822fcf3d-6d4c-4948-a71b-2d0b46db82a9/model
)'>
```

Deployment scorer properties

A deployment scorer has the following main properties:

- `api_base_url`: The base URL for all REST API interactions.
- `readyz_endpoint`: Endpoint for checking if the scorer is ready to receive requests.
- `capabilities_endpoint`: Endpoint that lists supported scorer capabilities. For more information, see View scorer capabilities.
- `schema_endpoint`: Endpoint that returns the input and output schema, including field names and types.
- `sample_request_endpoint`: Endpoint that returns a sample request payload with placeholder values.
- `scoring_endpoint`: Endpoint for submitting payloads to get model predictions.
- `media_scoring_endpoint`: Endpoint for scoring media inputs such as images, audio, or text.
- `contributions_endpoint`: Endpoint that returns feature contributions (Shapley values), if supported.

Access endpoints

You can use scorer methods to interact with deployment endpoints. Each method that sends a request to an endpoint accepts the following optional parameters:

- `auth_value`: `Optional[str] = None` – The deployment authorization value, such as a passphrase or access token. Required only if the endpoint is secured. For more information on security options, see Security options.
- `timeout`: `Optional[float] = 5` – The timeout in seconds for the HTTP request.

Note: For the `OIDC_AUTH` security option, use a valid access token as the authentication value. For example:

```
import h2o_authn

token_provider = h2o_authn.TokenProvider(
   refresh_token=...,
   client_id=...,
   token_endpoint_url=...,
)
auth_value = token_provider()
```

H2O MLOps client automatically tries to fetch the access token when none is provided.

View scorer state

To view the scorer's state:

Input:

```
scorer.state()
```

Output:

```
'Ready'
```

Check if the scorer is ready

To check if the scorer is ready to receive requests:

Input:

```
scorer.is_ready()
```

Output:

```
True
```

View scorer capabilities

To view the scorer's supported capabilities:

Input:

```
scorer.capabilities()
```

Output:

```
['SCORE_PREDICTION_INTERVAL', 'SCORE']
```

The available scorer capabilities are:

- `SCORE`: Enables standard predictions for structured data.
- `SCORE_PREDICTION_INTERVAL`: Allows returning prediction intervals.
- `CONTRIBUTION_ORIGINAL`: Supports computing Shapley values for original input features.
- `CONTRIBUTION_TRANSFORMED`: Supports computing Shapley values for transformed features.
- `MEDIA`: Accepts media inputs such as images, audio, or text for scoring.
- `TEST_TIME_AUGMENTATION`: Eligible to apply test-time augmentation and aggregate results.

For more information, see Advanced capabilities.

View schema

To view the input and output schema, including column names and their data types:

Input:

```
scorer.schema()
```

Output:

```
{
'id': '92aa5dee-5b6b-4a28-a19f-0156ae0bde34',
'schema': {
   'inputFields': [
     {'name': 'Origin', 'dataType': 'Str'},
     {'name': 'Dest', 'dataType': 'Str'},
     {'name': 'fDayofMonth', 'dataType': 'Str'},
     {'name': 'fYear', 'dataType': 'Str'},
     {'name': 'UniqueCarrier', 'dataType': 'Str'},
     {'name': 'fDayOfWeek', 'dataType': 'Str'},
     {'name': 'fMonth', 'dataType': 'Str'},
     {'name': 'IsDepDelayed', 'dataType': 'Str'}],
   'outputFields': [
     {'name': 'Distance', 'dataType': 'Float64'}]}
}
```

Note: The `id` in the response identifies the experiment associated with the scorer. This is especially helpful in multi-model deployments.

Example: View the experiments associated with the deployment.

Input:

```
deployment.experiments
```

Output:

```
   | name                  | uid
---+-----------------------+--------------------------------------
 0 | H2O3 MOJO experiment | 92aa5dee-5b6b-4a28-a19f-0156ae0bde34
```

The experiment ID matches the one returned by the scorer.

Generate a sample request

To generate a sample scoring payload with placeholder values:

Input:

```
scorer.sample_request()
```

Output:

```
{'fields': ['Origin',
 'Dest',
 'fDayofMonth',
 'fYear',
 'UniqueCarrier',
 'fDayOfWeek',
 'fMonth',
 'IsDepDelayed'],
'rows': [['text', 'text', 'text', 'text', 'text', 'text', 'text', 'text']]}
```

Create a payload

Here's an example of a manually defined payload with multiple rows:

```
payload = {
   "fields": [
       "Origin", "Dest", "fDayofMonth", "fYear", "UniqueCarrier", "fDayOfWeek", "fMonth", "IsDepDelayed"
   ],
   "rows": [
       ["text", "text", "text", "text", "text", "text", "text", "text"],
       ["text", "text", "text", "text", "text", "text", "text", "text"],
       ["text", "text", "text", "text", "text", "text", "text", "text"],
   ]
}
```

Score against the deployment

If the scorer supports the `SCORE` capability, you can send a payload to score the model:

Input:

```
scorer.score(payload=payload)
```

Output:

```
{'id': '92aa5dee-5b6b-4a28-a19f-0156ae0bde34',
'fields': ['Distance'],
'score': [['713.7770420135266'],
 ['713.7770420135266'],
 ['713.7770420135266']]}
```

Note: The `id` in the response identifies the experiment associated with the scorer. This is especially helpful in multi-model deployments.

Advanced capabilities

Prediction intervals

If the scorer supports the `SCORE_PREDICTION_INTERVAL` capability, you can request prediction intervals by adding the following flag to the payload:

```
payload["requestPredictionIntervals"] = True
```

Shapley values

If the scorer supports the `CONTRIBUTION_ORIGINAL` or `CONTRIBUTION_TRANSFORMED` capability, you can request feature contributions by setting the `requestShapleyValueType` field in the payload. If the scorer doesn't support the requested capability, it returns an error.

Example: `>python >payload["requestShapleyValueType"] = "ORIGINAL" >`

- `ORIGINAL`: Returns Shapley values for the original input features.
- `TRANSFORMED`: Returns Shapley values for the transformed features.

The scorer returns an array of rows containing the requested Shapley values, aligned with the `fields` specified in the request.

Note: If you only want feature contributions and not prediction results, use the `score_contributions()` method instead of `score()`. The input arguments are the same.

Media scoring

You can score media inputs, such as text, images, or audio, using either the `score()` or `score_media()` method.

Option 1: Use `score()` with base64-encoded content The `score()` method expects all media content as base64-encoded strings.

```
import base64

with open("/abs/path/to/file", "rb") as f:
 encoded = base64.b64encode(f.read()).decode()

payload["rows"] = [[encoded]]
```

Option 2: Use `score_media()` with file paths The `score_media()` method allows you to send raw media files directly by specifying the file paths.

```
payload["media_fields"] = ["file"]  # This must be a subset of payload["fields"]
payload["rows"] = [["file_name1"], ["file_name2"], ["file_name3"]]

scorer.score_media(
    payload=payload,
    file_paths=[
        "/abs/path/to/file1",
        "/abs/path/to/file2",
        "/abs/path/to/file3",
    ],
)
```

Batch scoring

This page guides you on how to use the H2O MLOps Python client for batch scoring.

For more information about batch scoring and the supported source and sink types, see Batch scoring.

Configure the input source

To list available source connectors, run:

```
mlops.batch_connectors.source_specs.list()
```

Use the following code to configure the input source:

Amazon S3

```
source = h2o_mlops.options.BatchSourceOptions(
   spec_uid="s3",
   config={
       "region": "us-west-2",
       "accessKeyID": credentials['AccessKeyId'],
       "secretAccessKey": credentials['SecretAccessKey'],
       "sessionToken": credentials['SessionToken'],
   },
   mime_type=h2o_mlops.types.MimeType.CSV,
   location="s3:///.csv",
)
```

Note: Public S3 buckets are also supported as an input sink. To read from the public S3 bucket, leave the access key and secret key fields empty. Only the input sink allows public S3 buckets.

GCP

```
source = h2o_mlops.options.BatchSourceOptions(
   spec_uid="gcp",
   config={
       "projectID": credentials['projectID'],
       "credentials": credentials['credentials'],
   },
   mime_type=h2o_mlops.types.MimeType.CSV,
   location="",
)
```

Azure

```
source = h2o_mlops.options.BatchSourceOptions(
   spec_uid="azure",
   config={
       "accountKey": credentials['accountKey'],
       "sasToken": credentials['sasToken'],
       "containerName": credentials['containerName']
   },
   mime_type=h2o_mlops.types.MimeType.CSV,
   location="https://.blob.core.windows.net/.csv",
)
```

MinIO

```
source = h2o_mlops.options.BatchSourceOptions(
   spec_uid="s3",
   config={
       "region": "us-west-2",
       "accessKeyID": credentials['AccessKeyId'],
       "secretAccessKey": credentials['SecretAccessKey'],
       "sessionToken": credentials['SessionToken'],
```

```
        "pathStyle": True,
        "endpoint": "https://s3.minio.location"
    },
    mime_type=h2o_mlops.types.MimeType.CSV,
    location="s3:///.csv",
)
```

JDBC

```
source = h2o_mlops.options.BatchSourceOptions(
    spec_uid="jdbc",
    config={
      "table": "table_with_data",
      "driver": "postgres",
      "numPartitions": 8,
      "lowerBound": "2023-01-01 00:00:00",
      "upperBound": "2024-01-01 00:00:00",
      "partitionColumn": "created_at",
      "secretParams": {
        "username": credentials["username"],
        "password": credentials["password"],
      }
    },
    mime_type=h2o_mlops.types.MimeType.JDBC,
    location="postgres://h2oai-postgresql.default:5432/db_name?user={{username}}&password={{password}}&ss
)
```

Configure the output location

To list available sink connectors, run:

```
mlops.batch_connectors.sink_specs.list()
```

This command returns schema details, supported paths, and MIME types.

Set up the output location where the batch scoring results will be stored:

Amazon S3

```
output_location = location="s3:////" + datetime.now().strftime("%Y%m%d-%H%M%S")
sink = h2o_mlops.options.BatchSinkOptions(
    spec_uid="s3",
    config={
        "region": "us-west-2",
        "accessKeyID": credentials['AccessKeyId'],
        "secretAccessKey": credentials['SecretAccessKey'],
        "sessionToken": credentials['SessionToken'],
    },
    mime_type=h2o_mlops.types.MimeType.JSONL,
    location=output_location,
)
```

GCP

```
output_location = location="" + datetime.now().strftime("%Y%m%d-%H%M%S")
sink = h2o_mlops.options.BatchSinkOptions(
    spec_uid="gcp",
    config={
        "projectID": credentials['projectID'],
        "credentials": credentials['credentials'],
    },
    mime_type=h2o_mlops.types.MimeType.JSONL,
    location=output_location,
)
```

Azure

```
output_location = location="https://.blob.core.windows.net//" + datetime.now().strftime("%Y%m%d-%H%M%S")
sink = h2o_mlops.options.BatchSinkOptions(
   spec_uid="azure",
   config={
       "accountKey": credentials['accountKey'],
       "sasToken": credentials['sasToken'],
       "containerName": credentials['containerName']
   },
   mime_type=h2o_mlops.types.MimeType.JSONL,
   location=output_location,
)
```

MinIO

```
output_location = location="s3:///" + datetime.now().strftime("%Y%m%d-%H%M%S")
sink = h2o_mlops.options.BatchSinkOptions(
   spec_uid="s3",
   config={
       "region": "us-west-2",
       "accessKeyID": credentials['AccessKeyId'],
       "secretAccessKey": credentials['SecretAccessKey'],
       "sessionToken": credentials['SessionToken'],
       "pathStyle": True,
       "endpoint": "https://s3.minio.location"
   },
   mime_type=h2o_mlops.types.MimeType.JSONL,
   location=output_location,
)
```

JDBC

```
sink = h2o_mlops.options.BatchSinkOptions(
   spec_uid="jdbc",
   config={
       "driver": "postgres",
       "table": "new_table",
       "secretParams": {
         "username": credentials["username"],
         "password": credentials["password"],
     }
   },
   mime_type=h2o_mlops.types.MimeType.JDBC,
   location="postgres://h2oai-postgresql.default:5432/db_name?user={{username}}&password={{password}}&ss
)
```

Create batch scoring job

First, retrieve the scoring runtime for the model:

```
scoring_runtime = model.experiment().scoring_runtimes[0]
```

To retrieve a list of available resource specifications for job creation, use:

```
mlops.batch_connectors.source_specs.list()
```

and

```
mlops.batch_connectors.sink_specs.list()
```

Create the batch scoring job:

```
job = workspace.batch_scoring_jobs.create(
   source=source,
```

```
    sink=sink,
    model=model,
    scoring_runtime=scoring_runtime,
    kubernetes_options=h2o_mlops.options.BatchKubernetesOptions(
        replicas=2,
        min_replicas=1,
    ),
    mini_batch_size=100, #number of rows sent per request during batch processing
    name="DEMO JOB",
)
```

Retrieve the job ID:

```
job.uid
```

Wait for job completion

During the execution of the following code, you can view the log output from both the scorer and the batch scoring job.

```
job.wait()
```

By default, this command will print logs while waiting. If you want to wait for job completion without printing any logs, use:

```
job.wait(logs=False)
```

List all jobs

```
workspace.batch_scoring_jobs.list()
```

Retrieve a job by ID

```
workspace.batch_scoring_jobs.get(uid=...)
```

Cancel a job

```
job.cancel()
```

By default, this command blocks until the job is fully canceled. If you want to cancel without waiting for completion, use:

```
job.cancel(wait=False)
```

Delete a job

```
job.delete()
```

Monitoring setup

This guide shows you how to configure monitoring for your deployment using the H2O MLOps Python client.

Follow the steps below to define monitored columns, optionally set up Kafka integration, deploy with monitoring enabled, or enable or disable monitoring after deployment.

Step 1: Define input and output columns

To enable monitoring in H2O MLOps, you must specify the input and output columns to monitor. You can do this in one of the following ways:

- Manual configuration
- Automatic configuration

Manual configuration

You can manually define the monitored columns using the `MonitoringOptions` class:

```
from h2o_mlops.options import (
    BaselineData,
    Column,
    MissingValues,
    MonitoringOptions,
    NumericalAggregate,
)
from h2o_mlops.types import ColumnLogicalType

options = MonitoringOptions(
    enabled=True,
    input_columns=[
      Column(
          name="age",
          logical_type=ColumnLogicalType.NUMERICAL,
      ),
    ],
    output_columns=[
      Column(
          name="quantity",
          logical_type=ColumnLogicalType.NUMERICAL,
          is_model_output=True,
      )
    ],
    baseline_data=[
        BaselineData(
            column_name="AGE",
            logical_type=ColumnLogicalType.NUMERICAL,
            numerical_aggregate=NumericalAggregate(
                bin_edges=[
                    float("-inf"),
                    22.0,
                    23.0,
                    25.0,
                    26.0,
                    28.0,
                    30.0,
                    31.0,
                    float("inf"),
                ],
                bin_count=[0, 1, 3, 1, 2, 2, 3, 3],
                mean_value=27.266666666666666,
```

```
                standard_deviation=3.2396354880199243,
                min_value=22.0,
                max_value=31.0,
                sum_value=409.0,
            ),
            categorical_aggregate=None,
            missing_values=MissingValues(row_count=0),
        ),
    ],
)
```

Automatic configuration

You can also configure monitoring automatically. This method can calculate the baseline using PySpark.

Note: PySpark is required for this step.

```
from h2o_mlops.types import ColumnLogicalType
from h2o_mlops.utils.monitoring import (
    Format,
    get_spark_session,
    prepare_monitoring_options_from_data_frame,
    read_source,
)

session = get_spark_session()

baseline_data_frame = read_source(
    spark=session,
    source_data="file:///datasets/categorical_data.csv",
    source_format=Format.CSV,
)

### User is able to override logical type for column for example ID column
logical_type_overrides = {
    "id": ColumnLogicalType.ID,
}

# Experiment is optional and base on schema in experiment code is able to discover proper types for moni
options = prepare_monitoring_options_from_data_frame(
    data_frame=baseline_data_frame,
    logical_type_overrides=logical_type_overrides,
    experiment=experiment,
)

options.enabled = True
```

Step 2: Optional: Kafka integration for raw scoring logs

You can enable the export of raw scoring request and response data to Kafka, if it is enabled in the environment. You can use a global topic or specify a custom topic per deployment.

```
options.kafka_topic = "test"
```

Step 3: Edit baseline and columns before deployment

You can modify the automatically detected baseline and monitored columns before deployment if the detection was inaccurate.

To modify the logical type of an existing column:

```
options.input_columns[0].logical_type = ColumnLogicalType.CATEGORICAL
```

To replace an entire column definition:

```
options.input_columns[0] = Column(
   name="width",
   logical_type=ColumnLogicalType.NUMERICAL,
)
```

Step 4: Configure monitoring for deployment

You can deploy a model with monitoring enabled, or enable or disable monitoring after deployment.

Deploy with monitoring enabled

To deploy with monitoring enabled:

```
deployment = workspace.deployments.create(
   name="demo-deployment",
   composition_options=[comp_opts],
   mode=DeploymentModeType.SINGLE_MODEL,
   monitoring_options=options,
   security_options=sec_opt,
)
```

Enable or disable monitoring after deployment

You can enable or disable monitoring after deployment as long as the monitored columns were provided. If they weren't, you must configure them first with the `monitoring_options` configuration.

To disable monitoring if it was already configured:

```
options = deployment.monitoring_options
options.enabled = False
(monitoring_options=options)
```

To enable monitoring when it wasn't configured at deployment time:

First, define the monitored columns using manual or automatic configuration.
For more information, see Step 1: Define input and output columns. Then:

```
options = deployment.monitoring_options
options.enabled = True
deployment.update(monitoring_options=options)
```


Python client migration guide

This guide compares the H2O MLOps Python client across versions. Each table shows how to perform an operation in the earlier version (left column) and in the later version (right column). Use these comparisons to update your code.

From v1.3.x to v1.4.x

Imports

v1.3.x

v1.4.x

Client creation

From v1.4.x onwards, support for creating the client using `gateway_url` and `token_provider` has been removed. Instead, you must use `refresh_token` and `h2o_cloud_url`.

v1.3.x

v1.4.x

```
token_provider = h2o_authn.TokenProvider(
    refresh_token=...,
    client_id=...,
    token_endpoint_url=...,
)
mlops = h2o_mlops.Client(
    gateway_url=...,
    token_provider=token_provider,
)
```

```
mlops = h2o_mlops.Client(
   h2o_cloud_url=<H2O_CLOUD_URL>,
   refresh_token=<REFRESH_TOKEN>,
)
```

Get allowed affinities and tolerations

v1.3.x

v1.4.x

```
      mlops.allowed_affinities
      ```
    </td>
    <td>
      ```python
      mlops.configs.allowed_k8s_affinities
      ```
    </td>
  </tr>
  <tr>
    <td>
      ```python
      mlops.allowed_tolerations
      ```
    </td>
    <td>
      ```python
      mlops.configs.allowed_k8s_tolerations
```

```
      ```
    </td>
  </tr>
</table>

Get the current user

<table>
<tr>
<th>v1.3.x</th>
<th>v1.4.x</th>
</tr>
<tr>
    <td>
    ```python
    mlops.get_user_info()
    ```
    </td>
    <td>
    ```python
    mlops.users.get_me()
    ```
    </td>
</tr>
<tr>
    <td>
    Returns the user's information as a Python dictionary.
    </td>
    <td>
    Returns the user's information as an `MLOpsUser` instance.
    </td>
</tr>
</table>

Access project-related services

In version 1.4.x, the concept of *projects* has been replaced by *workspaces*. Update your code by replacing

<table>
<tr>
<th>v1.3.x</th>
<th>v1.4.x</th>
</tr>
<tr>
<td>
```python
mlops.projects.<action>()
```
</td>
<td>
```
mlops.workspaces.<action>()
```
</td>
</tr>
</table>

Create and register an experiment into a model

The previous method of creating experiments and registering them with models is still supported.

v1.3.x

v1.4.x

```
experiment = project.experiments.create(
    data=..., name=...
```

```
)

model = project.models.create(name=...)

or

model = project.models.get(uid=...)

model.register(experiment=experiment)

model.register(
    experiment="/path/experiment.zip",
    name=...,
)

or

workspace.models.register(
    experiment="/path/experiment.zip",
    name=...,
)
```

Users can pass an instance of the `MLOpsExperiment` as well.

Note:

- When you link an experiment to a workspace from H2O Driverless AI, a new model version is automatically registered under the model that matches the experiment's name.
- If no matching model exists, a new model is created with the experiment name, and the experiment is registered as its first version.
- Therefore, you don't need to manually register experiments in MLOps. You can use the model directly.

Update an artifact's parent

v1.3.x

v1.4.x

```
artifact.update(
    parent_experiment=experiment,
)

artifact.update(
    parent_entity=experiment,
)
```

Get artifact's model-specific metadata (if applicable)

v1.3.x

v1.4.x

```
artifact.get_model_info()

artifact.model_info
```

Convert JSON artifact to a dictionary

v1.3.x

v1.4.x

```
artifact.to_dictionary()

artifact.to_dict()
```

Get the experiment associated with a model version

v1.3.x

v1.4.x

```
model.get_experiment(model_version=n)
```

```
model.experiment(model_version=n)
```

List scoring runtimes

The `experiment.scoring_artifact_types` property was removed in 1.4.x.

v1.3.x

v1.4.x

```
scoring_runtimes = mlops.runtimes.scoring.list(
    artifact_type=experiment.scoring_artifact_types[correct_index]
)
```

```
scoring_runtimes = experiment.scoring_runtimes
```

```
scoring_runtimes = mlops.runtimes.scoring.list(
    artifact_type=..., uid=...
)
```

```
scoring_runtimes = mlops.runtimes.scoring.list(
    artifact_type=..., runtime_uid=...
)
```

Note: When creating a deployment, instead of passing `scoring_runtimes[correct_index]`, you can use `mlops.runtimes.scoring.get(artifact_type=..., runtime_uid=...)` to get the `scoring_runtime`, if you already know the corresponding `artifact_type` and `runtime_uid`.

Create a deployment

v1.3.x

v1.4.x

```
project.deployments.create_single(
    name=...,
    model=...,
    scoring_runtime=...,
    security_options=options.SecurityOptions(
        passphrase=...,
        hashed_passphrase=...,
        disabled_security=...,
        oidc_token_auth=...,
    ),
)
```

```
workspace.deployments.create(
    name=...,
    composition_options=options.CompositionOptions(
        model=...,
        scoring_runtime=...,
    ),
    security_options=options.SecurityOptions(
        security_type=types.SecurityType.<TYPE>,
        passphrase=...,
    ),
)
```

Note: Starting in v1.4.x, when you create a deployment with hash-based security options, provide the passphrase directly. In earlier versions, you had to provide the hashed value instead.

Create a deployment with new model monitoring options

v1.3.x

v1.4.x

```
project.deployments.create_single(
    ...,
    monitoring_record_options=options.MonitoringRecordOptions(
        ...,
    ),
)
```

```
workspace.deployments.create(
    ...,
    monitoring_options=options.MonitoringOptions(
        ...,
    ),
)
```

Note: This is equivalent to how users created deployments with the old monitoring in the previous client. After the old monitoring was removed, this change was introduced. Note that the parameters accepted by `options.MonitoringOptions` differ from those used in the old monitoring.

Wait for deployment to become healthy

The previous method is still supported.

v1.3.x

v1.4.x

```
while not deployment.is_healthy():
    deployment.raise_for_failure()
    time.sleep(5)
```

```
deployment.wait_for_healthy()
```

Get deployment state

v1.3.x

v1.4.x

```
deployment.status()
```

```
deployment.state
```

```
deployment.is_healthy()
```

```
deployment.is_healthy
```

Update a deployment

v1.3.x

v1.4.x

```python
deployment.update_security_options(
    ...,
)
```



```python
deployment.update(
    security_options=options.SecurityOptions(
        ...,
    ),
    kubernetes_options=options.KubernetesOptions(
        ...,
    ),
    environment_variables={
        "KEY1": "VALUE1",
        "KEY2": "VALUE2",
    },
    monitoring_options=options.MonitoringOptions(
        ...,
    ),
)
```

```python
deployment.update_kubernetes_options(
    ...,
)
```

```python
deployment.set_environment_variables(
    environment_variables={
        "KEY1": "VALUE1",
        "KEY2": "VALUE2",
    },
)
```

```python
deployment.update_monitoring_options(
...,
)
```

Note: In v1.4.x, you can update multiple settings at once.

Access deployment scorer

v1.3.x

v1.4.x

You do not need to fetch the scorer.

```python
scorer = deployment.scorer

or

scorer = workspace.deployments.scorers(
    key=value,
)[index]
```

`deployment.scorer_api_base_url`

`scorer.api_base_url`

`deployment.url_for_capabilities`

```
scorer.capabilities_endpoint
```

```
deployment.url_for_schema
```

```
scorer.schema_endpoint
```

```
deployment.url_for_sample_request
```

```
scorer.sample_request_endpoint
```

```
deployment.url_for_scoring
```

```
scorer.scoring_endpoint
```

```
deployment.get_capabilities(...)
```

```
scorer.capabilities(...)
```

```
deployment.get_schema(...)
```

```
scorer.schema(...)
```

```
deployment.get_sample_request(...)
```

```
scorer.sample_request(...)
```

Score against a deployment

The previous method is still supported if the correct scoring endpoint URL is provided.

v1.3.x

v1.4.x

```
response = httpx.post(
    url=deployment.url_for_scoring,
    json=...,
)
```

```
response.json()
```

```
scorer.score(payload=...)
```

Kubernetes options for a batch scoring job

v1.3.x

v1.4.x

```
project.batch_scoring_jobs.create(
   ...,
   resource_spec=options.BatchKubernetesOptions(
       ...,
   ),
)
```

```
workspace.batch_scoring_jobs.create(
   ...,
   kubernetes_options=options.BatchKubernetesOptions(
       ...,
   ),
)
```

```
job.resource_spec
```

```
job.kubernetes_options
```

Get entity creator (if applicable)

v1.3.x

v1.4.x

```
entity.owner
```

```
entity.creator
```

View the complete `Table`

v1.3.x

v1.4.x

```
table
```

```
table.show(n=...)
```

Note: In version 1.4.x, a `Table` instance renders a nicely formatted view but displays only up to 50 rows by default.

From v1.2.x to v1.3.x

Removal of environments

v1.2.x

v1.3.x

```
environment = project.environments.get(uid=...)
```

You do not need to fetch the environment.

```
environment.deployments
```

```
project.deployments
```

```
environment.endpoints
```

```
project.endpoints
```

```
environment.allowed_affinities
```

```
mlops.allowed_affinities
```

```
environment.allowed_tolerations
```

```
mlops.allowed_tolerations
```

From v1.1.x to v1.2.x

There are no breaking changes.

From v1.0.x to v1.1.x

Minimal supported version

v1.0.x

v1.1.x

```
3.8
```

```
3.9
```

Create a deployment

v1.0.x

v1.1.x

```
project.deployments.create_single(
    name=...,
    model=...,
    scoring_runtime=...,
)
```

```
project.deployments.create_single(
    name=...,
    model=...,
    scoring_runtime=...,
    security_options=options.SecurityOptions(
        passphrase=...,
        hashed_passphrase=...,
        disabled_security=...,
        oidc_token_auth=...,
    ),
)
```

Note:

- The `security_options` field is no longer optional.
- To create a deployment with the `No Security` option:
  - For MLOps version 0.68.0 or later, set:
    `security_options = options.SecurityOptions(disabled_security=True)`
  - For MLOps versions earlier than 0.68.0, set:
    `security_options = options.SecurityOptions()`

H2O MLOps gRPC Gateway

The H2O MLOps gRPC Gateway provides a unified interface for interacting with the H2O MLOps platform's various services. This gateway serves as the entry point for all API operations, allowing you to manage models, deployments, monitoring, and other MLOps functionalities through a standardized API. To view the API specifications, open the H2O MLOps gRPC Gateway URL in a browser.

Note: If you're unsure how to access the H2O MLOps gRPC Gateway, contact your administrator.

API information

H2O MLOps uses gRPC (Google Remote Procedure Call) as its internal communication framework. The platform's services are exposed using gRPC Gateway. The services are not accessible directly, but instead the platform follows the API Gateway pattern and uses a gateway service to contact individual services.

API gateway health check

The API gateway includes a health check endpoint that is accessible at `/healthz`. Navigate to `<H2O_MLOPS_API_GATEWAY_URL>/healthz` for the health check.

Sample output:


```
{
    "deployment_server": {
        "message": "READY",
        "timestamp": "2025-06-16T17:52:32.855902715Z",
        "duration": 1260,
        "contiguousFailures": 0,
        "timeOfFirstFailure": null
    },
    "ingest": {
        "message": "READY",
        "timestamp": "2025-06-16T17:52:32.857480851Z",
        "duration": 951,
        "contiguousFailures": 0,
        "timeOfFirstFailure": null
    },
    "storage": {
        "message": "READY",
        "timestamp": "2025-06-16T17:52:32.858571077Z",
        "duration": 360,
        "contiguousFailures": 0,
        "timeOfFirstFailure": null
    }
}
```


Release notes

Version 1.0.0 (July 31, 2025)

This release marks a significant milestone in the evolution of H2O MLOps. It introduces the following major changes:

- The legacy Wave-based UI and Admin Analytics app have been replaced by the unified H2O AI Cloud user interface.
- A fully featured and user-friendly Python client is now available, replacing the previously generated client.
- The Deployer component has been rewritten to address previous architectural limitations.
- A new monitoring solution has been added, based on aggregated data and Apache Superset.

See the full release notes below for a complete list of new features, and fixes.

New Features

- Introduced a new native user interface that replaces the legacy H2O Admin Analytics Wave app and the H2O MLOps Wave app.
- Added a new model monitoring solution based on aggregated data and Apache Superset. To learn more, see Model monitoring.
- Added support for using CSV files **with headers** as sink (input) for batch scoring jobs. For configuration details, see Batch scoring.
- Enabled support for using a public S3 bucket as the source for batch scoring jobs.
- Integrated H2O MLOps with H2O AI Cloud's **Authz** service for authorization control.
- Integrated H2O MLOps with H2O AI Cloud's **Workspace** service. All projects have been migrated to Workspaces.
- Integrated H2O MLOps with H2O AI Cloud's **User** service.
- Added the ability to manually retry failed deployments.
- Added support for `AWS_MSK_IAM` and `SCRAM` Kafka authentication methods.
- Added support for forwarding input scoring data to customer-specific Kafka topics.
- Integrated H2O MLOps with the Audit Trail component. All API interactions from H2O MLOps components are captured and sent to the Audit Trail service for processing. Users can view the collected data in the Audit Trail UI.
- Upgraded the MOJO library used in runtimes to version 2.8.9.1.
- Added support for H2O Driverless AI runtimes:
  - 1.10.7.4
  - 1.10.7.5
  - 2.2.3
- Added support for specifying security contexts for all dynamically created pods, including batch scoring jobs, artifact fetchers, proxies, and runtimes.
- Added support for specifying affinity and tolerations in batch scoring jobs.
- Introduce competition statistics for batch scoring jobs.
- Introduce startup timeout option for batch scoring jobs and associated scorers.

Fixes

- Applied security fixes across all components.
- Allowed additional input fields beyond the expected schema in batch scoring jobs.
- Resolved Azure download timeout errors when Azure is used as the data source in batch scoring jobs.
- Added support for table names with special characters when using JDBC as the data source in batch scoring jobs.
- Fixed a segmentation fault that occurred during batch scoring when GCP was used as the data source.
- Corrected the supported MIME types for JDBC output sinks.
- Deployments are no longer processed sequentially.
- Prevented segmentation faults caused by concurrent hash generation in the security proxy component.
- Instead of failing whole batch scoring jobs when scoring fails, write the affected entry to error file and continue.

Python client v1.4.4

Fixes

- Ensured that an isolated `httpx` client with a limited connection pool is used internally for native scorer support.

Python client v1.4.3

New features

- Exposed configurable API timeout settings.

Fixes

- Prevented retrieval of large experiment and dataset metadata by allowing retrieval based on user inputs only.

Python client v1.4.2

Fixes

- Added support for accepting `job_start_timeout` when creating batch scoring jobs.

Python client v1.4.1

Fixes

- Added support for initializing the client by passing `token_provider` along with `h2o_cloud_url`.

Python client v1.4.0

New features

- Released the official H2O MLOps Python client (`h2o-mlops`), available on PyPI, which fully replaces the previously generated Python client. The generated client is no longer supported. For usage details, see H2O MLOps Python client.
- Added native support for accessing common scorer endpoints.

Removed features

- Removed `gateway_url` and `token_provider` from Client constructor. Users should use `h2o_cloud_url` instead.

Version 0.70.7 (May 30, 2025)

New Features

- Added support for the Scoring Runtime in H2O Driverless AI 2.1.0

Version 0.70.6 (May 29, 2025)

Fixes

- Python Client Unpinned Python package dependencies.
- [Wave UI] Fixed security vulnerabilities.

Version 0.70.5 (Apr 25, 2025)

Fixes

- Fixed an issue where batch scoring jobs failed when Azure was used as the input data source.

Version 0.70.4 (Apr 8, 2025)

Fixes

- [Deployer] Added missing service accounts to existing deployments during deployment updates.

Version 0.70.3 (Apr 3, 2025)

Fixes

- Fix deployer memory leak

Version 0.70.2 (Apr 3, 2025)

Fixes

- Ensured that the Driverless AI Scoring Pipeline does not run under the root user.
- Resolved regression that prevented scoring on the Java runtime by upgrading to the latest MOJO library.
- Fixed an issue where the Storage connection was incorrectly closed in the deployment server.

Version 0.70.1 (Mar 31, 2025)

New Features

- [Runtimes] Added support for H2O Driverless AI v2.0.0.

Fixes

- Added missing component versions to Go-based H2O MLOPs components.
- Batch Scoring Added missing security context to batch scoring job pods.
- [Security] Fixed newly discovered vulnerabilities.
- [Deployer] Added missing access token for the `readArtifactAsAdmin` operation.
- [Helm] Added configuration to refresh intervals for JWKS keys in service-to-service authentication. This ensures timely updates, regardless of Cache-Control headers.

Version 0.70.0 (Mar 13, 2025)

New features

- Added support for Workload Identity in all MLOPs components.
- Enabled IAM support across all MLOps components.
- Replaced SPIFFE with service accounts for service-to-service authentication.
- Removed the custom TLS implementation. Users are encouraged to use a service mesh, such as Istio, to secure in-cluster communication.
- Introduced native Batch Scoring implementation. Users can access this capability through the Python client and the new user interface.
- Launched a new user interface that runs alongside the existing one for the upcoming release.
- [Helm] Moved global CORS configuration to Helm.

Fixes

- Fixed vulnerabilities across all MLOps components.
- [Storage] Updated to return a `Not Found` gRPC error code when a dataset cannot be found in the storage database.
- [Monitor Proxy] Updated to pass secrets instead of plain text for Kafka credentials.
- [Helm] Fixed automatic cleanup of the updater job.
- [Wave UI] Prevented redirection to projects while filling out the **Create deployment** form.

Version 0.69.7 (Feb 17, 2025)

Fixes

- [Deployer] Resolved an issue where resource limit specifications were not correctly applied to runtime processors.

Version 0.69.6 (Feb 13, 2025)

Fixes

- [Security] Applied security patches from the latest major release.
- [Deployer] Fixed an issue by adding the missing volume mount for Kafka TLS-enabled deployments.

Version 0.69.5 (Feb 6, 2025)

Fixes

- [Wave App] Prevented redirection to `#projects` while filling out the `create deployment` form.

Version 0.69.4 (Jan 21, 2025)

Fixes

- [Helm Chart] Updated the InfluxDB network policy to allow connections from pods with any of the required labels.

Version 0.69.3 (Jan 17, 2025)

Fixes

- [Wave UI] Fixed an issue where Driverless AI version 1.11.1.1 was incorrectly displayed as 1.11.1 in the UI.

Version 0.69.2 (Jan 14, 2025)

This release includes new features and fixes.

New features

- [Deployment Updater] Added functionality to update the image repository during deployment update jobs.

Fixes

- [Deployer] Fixed `CVE-2023-3635`.

Version 0.69.1 (Jan 9, 2025)

This release includes a new feature.

New features

- [Runtimes] Added support for the Driverless AI runtime version 1.11.1.1.

Version 0.69.0 (Dec 19, 2024)

This release includes new features and fixes.

New features

- [Runtimes] Added support for MLflow Model Scorer runtime for Python 3.11 and 3.12, and Dynamic MLflow Model Scorer runtime for Python 3.12.
- [Runtimes] Added support for H2O Driverless AI runtime version 1.10.7.3.
- [Runtimes] Exposed Kubernetes readiness probe on deployments.
- [UI] Replaced bcrypt with PBKDF2 hashing when creating deployments with the `Passphrase (Stored Hashed)` security option.
- [Deployer] Introduced PBKDF2-based passphrase hashing for improved security.
- [Deployer] Added support for Generic Ephemeral Volumes in the Runtimes.
- [Deployer] Introduced `/readyz` readiness probe endpoint for dynamically deployed runtimes.
- [Deployer] Introduced a pod disruption budget for enhanced stability.
- [Helm] Enabled JVM config passing to the monitoring proxy.
- [Helm] Added support for configuring limits and JVM settings in the deployer.
- [Helm] Defined `MLOPS_WAVE_APP_URL` as an environment variable for better configuration.

Fixes

- [UI] Ensured UI accessibility even when listing deployments fails.
- [UI] Fixed the issue where signing in from the access denied page resulted in an `Missing parameters: id_token_hint` error.
- [Monitor Proxy] Stopped sending `TransactionTransmission` events to downstream transmitters when `enableTransaction` is `false`.
- [Storage] Removed storage cleanup cron job and implemented it within a thread of storage itself.
- [Helm] Ensured proper RBAC configuration when multiple groups are specified.
- [Helm] Removed legacy `LOCAL` and `MIGRATE` mode code.

- [Runtimes] Fixed memory leak in MOJO2 runtimes by upgrading the internal MOJO2 library.
- [Deployer] Ensured that stale deployments will be redeployed.
- [Deployer] Skipped routing migration in cases of errors not related to deployment migration.
- [Deployer] Used response header modifier instead of request header modifier for CORS.
- [Deployer] Added configurable Kubernetes client timeout for better performance and reliability.

Python client v1.2.0

New features

- Added support for SSL settings via the Client constructor.
- Added support for PBKDF2 hashed security option.

Fixes

- Added `ValueError` for missing or invalid protocol in the gateway URL.
- Configured SSL settings for the functions `get_capabilities`, `get_sample_request`, and `get_schema` since they access deployment endpoints.

Version 0.68.0 (Nov 05, 2024)

This release includes new features and fixes.

New features

- [UI] Enabled model and model version deletion.
- [UI] Enabled to use the default deployment security option from the backend.
- [UI] Added support for H2O Driverless AI runtime versions 1.10.7.2 and 1.11.1.
- [Storage] Storage only supports blob storage from this release onwards. A one-time migrator job was introduced to migrate all the storage data from K8S PVC to blob storage to support seamless upgrades for users.
- [Telemetry] The MLOps-Telemetry component is no longer running as a cron job; it is now a long-running microservice that publishes event data at scheduled intervals.
- [Helm] MLOps storage can be configured to use blob storages from any of the 3 main clouds AWS, Azure and GCP. Minio is also supported for on-premise installations.
- [Helm] Added `H2O_SCORER_MODEL_LOADING_MODE` set to "subprocess" across all MLOps Python-based runtimes.
- [Helm] Introduced a migration job for transferring persistent storage to cloud platforms, now supporting Minio and Azure Blob.
- [Helm] Introduced a `SCHEDULER_INTERVAL_SECONDS` environment variable to configure the interval of mlops-telemetry events publishing.
- [Deployer] Introduced Vertical Pod Autoscaling (VPA) support.
- [Deployer] Exposed easy access to the security options available in the cluster.
- [Deployer] Restructured environment security options:
  - Activated security options list
  - Configurable default security option
- [Deployer] Introduced the **No-Security** option.

Fixes

- [UI] Resolved an error occurring when attempting to view experiment details for experiments with missing metadata.
- [UI] Made the maximum selectable count for deployment replicas configurable.
- [UI] Removed support for MLflow Model Scorer and Dynamic Model Scorers for Python 3.8.
- [UI] Removed support for HT Flexible Runtimes for Python 3.8, including both GPU and CPU variants.
- [gRPC Gateway] Updated `/healthz` to return a 200 status if at least one health check passes, fixing an issue where the gateway would restart if any service was unhealthy.
- [Helm] Removed Python 3.8 support for HT and MLFlow runtimes.
- [Helm] Removed the `EnableUserExternalIDUpdate` environment variable from storage for simpler configuration.
- [Helm] Added a `-job` suffix to the `app.kubernetes.io/`component label for the monitoring backend job to improve component labeling.
- [Helm] Updated rclone configurations to enhance compatibility with Google Cloud Storage (GCS).
- [Helm] Set the telemetry service's replica count to one to optimize resource usage.
- [Helm] Changed the telemetry scheduler's default interval to 300 seconds for more efficient scheduling.

- [Storage] `IDP_ID` (i.e. keycloak/ Okta ID) is now used as the primary key for the Users table in MLOps Storage. The username is also not a unique field anymore. Existing user data will be migrated accordingly by the Storage itself when it's spinning up. [Deployer] Only "internal" grpc status are now logged at the ERROR level.

Python client v1.1.2

New features

- Introduced two SSL settings (`verify_ssl` and `ssl_cacert`) that can be configured via the Client Constructor to improve certificate security.

Python client v1.1.0

New features

- Introduced a timeout parameter (default: 5 seconds) for MLOpsScoringDeployment's methods: `get_capabilities`, `get_sample_request`, and `get_schema`.
- Added support for creating deployment with token-based authentication as a security option.
- Enabled model deletion.
- Enabled the option to unregister an experiment from a model.
- Introduced the `disabled_security` option to manage deployments with **No-Security**.

Fixes

- Improved handling of missing deployment attributes (security and monitor) in backend responses.
- Upgraded the minimum supported Python version to 3.9.

Python client v1.0.1

Fixes

- Improved handling of missing deployment attributes (security and monitor) in backend responses.

Version 0.67.4 (Oct 10, 2024)

This release includes various fixes.

Fixes

- [Helm] Gateway creation is now skipped when `Values.gatewayApi.create` is set to false.
- [Helm] You can now specify extra ingress for Influx.

Version 0.67.3 (Oct 01, 2024)

This release includes new features and fixes.

New features

- [Runtimes] Added support for the Driverless AI 1.11.1 Python scoring pipeline.

Fixes

- [Security] Fixed critical vulnerabilities on Java-based rest scorer and monitoring proxy.
- [Helm] Ensure that registry specification on each image has higher priority over the global image registry configuration.

Version 0.67.2 (Sep 19, 2024)

This release includes new features and fixes.

New features

- [Runtimes] Added support for the Driverless AI 1.10.7.2 Python scoring pipeline.

Fixes

- [Helm] Removed hard-coded dev/vorvan prefix.
- [Helm] Influx network policy was missing a specific label which lead to cleanup job not running.

Version 0.67.1 (Sep 13, 2024)

This release includes various fixes.

Fixes

- [Monitoring Backend] Updated Dockerfile to use numerical user ID, preventing false warnings in systems that check for root access.
- [Drift] Fixed an issue where the worker image could not find the `datatable` dependency.

Version 0.67.0 (Sep 02, 2024)

This release includes various vulnerability fixes.

New features

- [Monitor Proxy] Per project monitoring data retention period can be set for Influx DB during the MLOps installation or upgrade.
- [UI] Added the functionality to log out from the Wave app.
- [UI] Added support for new HT Flexible Runtimes for Python `3.10`, including GPU and CPU variants.
- [UI] Added support for DAI runtime versions `1.10.6.3`, `1.10.7.1`, and `1.11.0`.
- [UI] Added support for MLflow Model Scorer and Dynamic Model Scorers for Python `3.10` and `3.11`.
- [Deployer] Added support for token based authentication for deployments.
- [Runtimes] Added support for DAI runtime versions `1.10.6.3`, `1.10.7.1`, and `1.11.0`.
- [Runtimes] Added support for new HT Flexible Runtimes for Python `3.10`.
- [Helm Chart] Added component configuration support for applying tolerations, node selectors, and affinity settings to cron jobs.
- [Helm Chart] Added CA certificate support to the API Gateway deployment.
- [Helm Chart] Replaced Ambassador with Gateway API due to the removal of Emissary.

Fixes

- [UI] Removed the functionality for importing models from external model repositories.
- [UI] Removed the ability to upload experiments as serialised Python (`.pkl`/`.pickle`) files.
- [UI] Disallowed the creation of tags with commas.
- [UI] Reduced the timeout for notification bars.
- [UI] Fixed the issue where a red cross appeared when registering a model shortly after creating an experiment.
- [UI] Removed support for DAI Python runtimes for `1.10.4.3` and older versions.
- [UI] Removed support for MLFlow Model Scorer for Python `3.6` and Python `3.7`.
- [Runtimes] Removed support for DAI Python runtimes for `1.10.4.3` and older versions.
- [Runtimes] Fix critical vulnerabilities in all runtimes except DAI Python based one.
- [Runtimes] Fix critical and high vulnerabilities in rest scorer.
- [Helm Chart] Corrected the nodeSelector YAML formatting.
- [Helm Chart] Renamed environment variable `STORAGE_URL` to `API_GATEWAY_URL` in the Wave app.
- [Helm Chart] Updated `H2O_WAVE_POST_REDIRECT_URL` to resolve "Page Not Found" errors when logging out from the Wave app.
- [Helm Chart] Updated the Wave app secret `H2O_WAVE_OIDC_END_SESSION_URL` for improved logout functionality.
- [Helm Chart] The `enable_user_externalid_update` setting is now configurable.
- [Helm Chart] Exposed resource requests and limits for monitoring-drift, model-ingest, and api-gateway components.

Python client v1.0.0

Removed features

- The `external_registry` package has been removed.

Version 0.66.1

This release includes various vulnerability fixes.

New features

- Released Base Python Scorer v1.2.0 (BYOM).
- Released Python based runtimes v1.2.0 (BYOM).
- Released HT runtime v1.2.0 (BYOM).
- Released MLflow runtime v1.2.0 (BYOM).

Version 0.66.0 (June 04, 2024)

This release includes new features, improvements, bug fixes, and security improvements.

Announcement This version of H2O MLOps adds optional role-based access control (RBAC). This feature relies on two RBAC configurations: one for the front end (FE) and the other for the back end (BE). **When using RBAC, both configurations must be set up identically to ensure proper functionality and a seamless user experience.**

The following is an illustrative configuration for both FE and BE RBAC. Note that in this example, it is assumed that "admin" is included in the user access token groups claim. However, it is important to customize this configuration based on your specific requirements at the time of deployment.

```
apiGateway:
  config:
    # -- Log verbosity.
    logLevel: "debug"
    authorization:
      # -- Whether authorization is enabled.
      enabled: true
      # -- JWT claim key which contains the role/group information.
      userJwtClaim: "groups"
      # -- List of role/group values that should have access to MLOps API Gateway as an array.
      allowedUserRoles: ["admin"]
waveApp:
  authorization:
    # -- Whether authorization is enabled.
    enabled: true
    # -- JWT claim key which contains the role/group information.
    userJwtClaim: "groups"
    # -- Comma separated list of role/group values that should have access to MLOps FE.
    allowedUserRoles: "admin"
    # -- Separator character for role/group values in JWT claim.
    userRoleValueSeparator: ",
```

New features

- Added optional role-based access control (RBAC). You can now limit access to H2O MLOps APIs to users with specific roles.
- Released Base Python Scorer v1.1.1 (BYOM).
- Released Python-based runtimes v0.6.5 (BYOM).
- Released HT runtime v0.6.5 (BYOM).
- Released MLflow runtime v0.6.5 (BYOM).
- Exposed authorization header with bear token through Env Vars in base scorer (BYOM).
- You can now set configurable read time out in scorer proxy (Monitor Proxy)
- Added deployer configurable monitor proxy timeout ()Deployer)

- Upgraded base images to Java 17 eclipse-temurin:17.0.10_7-jre (Deployer)

Bug fixes

- Handle error while fetching additional details of the events related to an experiment after deleting a deployment.

Version 0.65.1 (May 25, 2024)

This release is a minor release on top of v0.65.0 with the storage and telemetry features rebuilt using the latest zlib.

Python client v0.65.1a3

Fixes

- vLLM Configuration artifacts not being uploadable because of a missing target column.

  Note: vLLM Configuration artifacts are a preferred alternative to using MLflow to create vLLM artifacts.

Python client v0.65.1a2

Fixes

- Failures when a deployment is missing Kubernetes options.

Python client v0.65.1a1

New features

- Ability to disable storage of scored data when monitoring is enabled. For more information, see Monitoring options.

Changes

- Use H2O MLOps 0.65.1 backend.

Version 0.65.0 (May 08, 2024)

This release includes new features, improvements, bug fixes, and security improvements.

New features

- Added the capability to disable model monitoring features when deploying H2O MLOps. Set the `monitoring_enabled` installation parameter to `false` to disable the following:
  - monitoring service
  - monitoring task
  - drift worker
  - drift trigger
  - InfluxDB
  - RabbitMQ
  - **Note:** that setting the `monitoring_enabled` parameter to `false` also disables health checks for the monitoring backend.
- Support for FEDRAMP compliance.
- Added an option to restrict model imports to specific types.
- Added support for vLLM config model types.
- When creating a deployment, added a deployment multi-issuer token security option. For more information, see Endpoint security.
- The `ListProjects` API now returns all projects for admin users.

- You can now set a specific timeout only for external registry import API.
- You can now upload LLM experiments using the MLOps Wave app.
- Added support for Authz user format.

Improvements

- By default, the model monitoring page is now sorted by deployment name.
- You can now configure the supported experiment types for the upload experiment flow.

Bug fixes

- Fixed incorrect sorting in the `listMonitoredDeployments` API response.

Version 0.64.0 (April 08, 2024)

New features

- MLflow Dynamic Runtime: Added support for Python 3.10.
- Added support for DAI 1.10.7 and 1.10.6.2 runtimes.
- Upgraded Rest scorer to Spring Boot 3 (1.2.0).
- Added vLLM runtime support.
- When creating a new deployment, added an option to disable monitoring for the deployment.
- Added validation for experiment file uploading.
- Extended scoring API with new endpoint `/model/media-score` to support uploading multiple media files.
- The H2O Hydrogen Torch runtime is now supported with the ability to score image and audio files against the new endpoint `/model/media-score`.
- The project page now includes an Events tab with pagination, search, and sorting.
- You can now delete experiments.
- Added pagination, search, sorting, and filtering by Tag on the Experiments page.
- The Create Deployment workflow now automatically populates K8s limits and requests with the suggested default settings.
- The deployment state is now updated dynamically on the Deployments page.
- Additional details about error deployment states are now displayed in the MLOps UI.
- You can now update and delete tags. Note that tags can only be deleted if they are not associated with any entity.

Improvements

- You can now edit the GPU request/limit fields.
- When creating a deployment, improved automatic population of the Kubernetes resource requests and limits fields in the UI based on the selected runtime and artifact type.
- H2O Driverless AI versions are now automatically identified when DAI models are uploaded through the Wave app or Python client.
- The deployment overview now displays additional details about errored deployment states.

Python client v0.64.0a2

New features

- Ability to set environment variables in the scoring runtime of a new or existing deployment.

Python client v0.64.0a1

Changes

- Use H2O MLOps 0.64.0 backend.

Version 0.62.5

In addition to the changes included in the 0.64.0 release, this release includes the following changes:

Improvements

- The Deployer API now lets you create and update deployment settings related to what monitoring data you want to save. For example:

```
deployment.monitor_disable = True
deployment.store_scoring_transaction_disable = True
deployment = mlops_client.deployer.deployment.update_model_deployment(
    mlops.DeployUpdateModelDeploymentRequest(deployment=deployment)
)
```

Changes

- Model Monitoring is now disabled by default for new deployments.

Known issues

- Monitoring settings can only be modified using the Python client, regardless of whether they were initially set via the UI or Python client.
- H2O MLOps version 0.62.5 cannot be upgraded to version 0.64.0. Upgrades from this version can only be made to version 0.65.0 and later.

Version 0.62.4

Improvements

- Various security improvements to address XSS security issues.

Version 0.62.1

New features

- You can now use the `ListExperiments` API to filter experiments by status (ACTIVE, DELETED). By default, the API returns ACTIVE experiments.

Improvements

- Added support for the DAI 1.10.6.1 runtime.
- Added pagination support in the Experiments page.

Bug fixes

- Fixed an issue where uploading large artifacts (above 40GB) resulted in an error.
- Fixed an issue where a registered model with the same name as a deleted model could not be created.

Announcements

- The URL link to the legacy H2O MLOps app has been removed.
- The legacy H2O MLOps app is no longer installed by default.

Python client v0.62.1a7

Fixes

- `"latest"` model version specifier didn't always retrieve the last model version created. This could cause deployments to use the wrong model version.

Python client v0.62.1a6

New features

- Ability to download experiment artifacts. For more information, see Experiment artifacts tutorial.
- Ability to override experiment artifact `mime_type` when adding a new artifact. For more information, see Experiment artifacts tutorial.

Python client v0.62.1a5

New features

- Metadata property for experiments.

Python client v0.62.1a4

New features

- Owner attribute for deployments, experiments, models, and projects.
- Ability to view deployment Kubernetes options (including requests, limits, affinity, and toleration).
- Ability to update deployment Kubernetes options (including scaling deployment down to zero resource usage).
- Ability to view deployment security options.
- Ability to update deployment security options (including changing passphrase for existing deployments).
- Ability to enable/disable monitoring for new and existing deployments.

Changes

- "UNHEALTY" status typo corrected to "UNHEALTHY".

Python client v0.62.1a3

New features

- Support for experiment artifacts.

Changes

- `experiment.artifact_types` renamed to `experiments.scoring_artifact_types`.

Fixes

- List methods not returning over 100 entries.

Python client v0.62.1a2

New features

- Support for experiment comments.
- Support for experiment tags.
- Integration of https://github.com/h2oai/cloud-discovery-py.

Python client v0.62.1a1

Changes

- Use MLOps 0.62.1 backend.

Version 0.62.0 (September 10, 2023)

New features

- For GPU-enabled model deployments, you can now set the appropriate Kubernetes (K8s) requests and limits by clicking the GPU Deployment toggle when creating a deployment. For more information, see Deploy a model and Kubernetes options.
- You can now create and assign experiment tags within a project. For more information, see Project page tabs.
- You can now edit the names and tags of experiments. For more information, see Project page tabs.

Improvements

- **Viewing projects:**
  - The default view when viewing projects has been changed from the grid view to the list view.
  - The Project ID of each project is now displayed in the list view.
  - The list view now features pagination, sorting, and search capabilities.
  - You can now search for a project by project name.
  - You can now sort the list of projects by time of creation and last modified time.
- **Project list view actions:** You can now view, share, and delete projects from the project list view. For more information, see List view actions.
- Improved UI for project sharing.
- Enhanced the Deployment Overview window to include Kubernetes settings and deployed model details across all deployment modes. For more information, see Understand the Deployment Overview window.
- **Python client:**
  - You can now enable or disable model monitoring for a deployment.
  - You can now update the deployment security option or password.
  - You can now delete experiments.
  - You can now delete Registered Model and Model Version.
- **Scoring:**
- Prediction intervals are now supported for MOJOs and Driverless AI Python scoring pipelines. Prediction intervals provide a range within which the true value is expected to fall with a certain level of confidence. You can check if prediction intervals are supported by using the `https://model.{domain}/{deployment}/capabilities` endpoint.
- Added a new MLflow Dynamic Runtime to dynamically resolve the various model dependencies in your MLflow model. For more information, see MLflow Dynamic Runtime.

Bug fixes

- Fixed an issue where the passphrase field could not be edited when creating a secured deployment.
- Fixed an issue that affected accurate sorting when using the sort by date functionality.

Version 0.61.1 (June 25, 2023)

Improvements

- Added support for Kubernetes 1.25.
- Added support for H2O Driverless AI version 1.10.5.

Bug fixes

- Various bug fixes to the deployment pipeline, monitoring, and drift detection.

Python client v0.61.1a3

Changes

- `MLOpsClient` class renamed to `Client`.
- `_mlops_backend` attribute renamed to `_backend`.

Version 0.61.0 (May 24, 2023)

New features

- You can now create A/B Test and Champion/Challenger deployments through the UI. For more information, see Deploy a model.
- You can now create and view configurable scoring endpoints through the UI. For more information, see Configure scoring endpoint.
- Concurrent Scoring Requests are now supported for Python-based Scorers. Scoring times for for C++ MOJO, Scoring Pipeline, and MLflow types now support parallelization with the default degree of parallelization set to 2. This can be changed with the environment variable `H2O_SCORER_WORKERS`. For more details, contact your H2O representative.

Improvements

- Added support for H2O-3 MLflow Flavors and importing of MLflow-wrapped H2O-3 models.

Version 0.60.1 (April 02, 2023)

New features

- Introduced a feature flag to enable the import third-party experiments (pickled experiments) flow with Conda. If you require Conda or third-party pickle import, this flag needs to be set at the time H2O MLOps is installed to continue using pickled experiments. For more information about enabling this feature flag when installing or upgrading H2O MLOps, contact support@h2o.ai.

Improvements

- You can now search for users by username when sharing a project with another user. You can now also sort the user list in alphabetical order.
- In the model monitoring feature summary table, the figures are now displayed only up to three decimals places.
- When no deployment name is present for the deployment, the deployment ID is now displayed as the name.
- A blocking error page is now shown to the user in case Keycloak is unavailable.
- Date and time are now both displayed for model monitoring predictions over time plot.
- Storage Telemetry now includes the additional fields Deployment Name and model version number.

Bug fixes

- Fixed a bug that caused experiments to fail during upload / ingestion.
- All dialogs in the UI can now can be closed with the escape key.
- Fixed a bug where drift was not previously calculated when a feature was determined to be a datetime type and the date time format was missing.

Version 0.59.1

Improvements

- Added support for the DAI 1.10.4.3 runtime.

Version 0.59.0 (February 12, 2023)

New features

- Storage telemetry: MLOps can now send analytical data related to storage operations to the telemetry server.
- Scoring telemetry: MLOps Scoring now sends scoring-related data to the telemetry server.
- Static scoring endpoints: You are now able to define and update a persistent URL that points to a particular MLOps deployment.
- **Deployment:**
  - Deployed scoring applications now set additional Kubernetes annotations.
  - Deployment APIs now return more accurate and useful gRPC status codes and error messages.
  - You can now download Kubernetes logs from deployments in the MLOps Wave App and MLOps API.

Improvements

- Upgraded the `h2o-wave` version to 0.24.1.
- Added support for the DAI 1.10.4.1 and DAI 1.10.4.2 runtimes.
- Updated the Python client.
- Added a cleanup task for files uploaded to the wave server.
- Updated the eScorer URL of the wave app deployment pipeline
- Added a new Kubernetes limit for the Hydrogen Torch runtime in the deployment creation flow.

Bug fixes

- Removed the custom implementation for the token provider.
- Removed the `artifact-id` from the `DeployDeploymentComposition` endpoint.
- Updated the packages in the base docker image.
- Fixed an issue related to displaying the session timeout page for deployment overview, view monitoring, and monitoring homepage.
- Fixed an issue where the drift detection trigger blocked the other calculations by adding timeout support to the InfluxDB client in trigger and worker.

Version 0.58.0 (December 15, 2022)

Improvements

- Added support for Kubernetes 1.23.
- Added support for H2O-3 MOJOs up to version `3.38.0.3`.
- Added support for linking and deploying H2O Driverless AI unsupervised models.
- Added support for scoring H2O Driverless AI MOJOs with the C++ MOJO runtime.
- Added support for TTA for H2O Driverless AI Python pipelines.
- Shapley values can now be calculated for H2O Driverless AI Python pipelines and MOJOs.
- Datetime columns for H2O Driverless AI models are now automatically detected.
- Fixed an issue where the Driverless AI Python Pipeline scorer occasionally restarted randomly.
- Updated ML Python packages in the standard Python scorer to support a wider range of custom user models.
- **BYOM scoring:**
  - Extended the Python scoring library to conform to v1.2.0 of the Scoring API.
  - Unexpected input fields are now ignored when performing scoring.
  - Introduced a feature that lets scorers override sample requests.
  - Implemented an experimental API for image and file scoring.
  - Replaced time-based handling of signals coming from Driverless AI scoring processes with static handling.
  - Added a Driverless AI MOJO Pipeline artifact processor image.
  - Added an H2O-3 artifact processor image.
  - Updated the DAI pipeline processor dependencies to address security vulnerabilities.

Documentation

- Added a page that describes support for Test Time Augmentation (TTA) in H2O MLOps.
- Added several new Python client examples.
- Updated the page on Deploying a model.

Version 0.57.3 (November 16, 2022)

New features

- You can now view monitoring dashboards for deployments directly through H2O MLOps. For more information, see Model monitoring.

Version 0.57.2 (August 01, 2022)

New features

- When browsing the MLflow directory, you can now search for specific MLflow models by name. Note that this search functionality is case sensitive, and that the model name can contain only letters, numbers, spaces, hyphens, and underscores up to 100 characters.
- When browsing the MLflow directory, the list of MLflow models is now organized into pages. You can specify the number of models listed on each page.

Bug fixes

- Fixed an issue where MLflow models could not be reimported.

Version 0.56.1 (May 16, 2022)

New features

- Azure access tokens can now be retrieved through H2O MLOps.

Improvements

- When creating a deployment, only deployable artifacts are now shown.
- Added **Driverless AI (DAI)** 1.10.2 and 1.10.3 as recognized versions of DAI for matching with DAI runtimes.
- H2O MLOps now displays either a success or error message when attempting to create a deployment.
- The process of linking models to an experiment is now simpler.
- H2O MLOps can now handle large text fields.
- Updated the H2O MLOps logo.
- Removed scroll bars in overview UI pages.

Bug fixes

- Fixed an issue that caused alignment issues between project cards.
- Underscores can now be used at the beginning of project names.
- Fixed an issue that caused H2O MLOps to crash when the deployer was restarted.
- Fixed an issue related to adding new comments to an experiment.

Version 0.56.0 (April 18, 2022)

New features

- Added support for batch scoring. For more information, see Deploying a model.
- Added support for H2O-3 MOJOs up to version `3.32.0.2`.

Version 0.55.0 (March 31, 2022)

New features

- Added support for integration with MLflow Model Registry.
- Admin users can now monitor H2O MLOps usage within their organization with Admin Analytics.

Documentation

- Added a new page on enabling third-party model management integration.
- Added a new section on adding experiments from MLflow Model Registry.

Version 0.54.1 (March 08, 2022)

New features

- H2O Driverless AI (DAI) 1.10.2 is now supported. Experiments trained in DAI 1.10.2 can now be managed and deployed by H2O MLOps.

Version 0.54.0 (February 03, 2022)

- New MLOps user interface.
- **Pickle model support**: Python serialized models in Pickle format can now be imported directly into MLOps. This means that you can use your third-party models without relying on packagers like MLflow.
- **Model Registry and Model Versioning**: You can now register your experiments using MLOps Model Registry and group new versions of a model using MLOps Model Versioning. Note that an experiment must first be registered in the MLOps Model Registry before being deployed.

Version 0.53.0 (January 18, 2022)

Notice

- Updated required MLOps Terraform providers to benefit from bug fixes and expanded support for setting Kubernetes options. Note that upgrading MLOps with the updated Terraform templates results in Terraform generating a lengthy state file differential to review.

Improvements

- Added three new MOJO scorers to the default MLOps configuration. Each of these scorers provide support for returning Shapley values along with model scoring.
- By default, all MLOps components now run as non-root users.
- By default, all third-party services deployed by MLOps except for RabbitMQ and Traefik run as non-root by default.
- Added support for setting a subset of Kubernetes Security Context options for any BYOM image.
- Exposed many new MLOps configuration fields as Terraform variables.
- Extended model scorers' capabilities to recover from connection and timeout issues.
- Exposed option to set arbitrary Kubernetes resource requests and limits for MLOps model deployment.
- Exposed option to set number of desired Kubernetes pods for model deployments.
- Fixed an issue where deployments reported incorrect last modified timestamps.
- Added `name` and `description` fields to model deployment API objects, allowing deployments to be user-labelled.
- Fixed an issue where MLOps' Deployer complained if certain BYOM configurations were missing. Defaults are now correctly applied unless overridden.
- Fixed an issue where one of Deployer's APIs was not exposed with the MLOps API. Known and available deployment environments (that is, Kubernetes clusters) may now be queried with the MLOps API.
- BYOM containers can now have their log levels be globally configurable.
- Exposed a number of configuration fields for bundled third-party services.
- Reduced factor of Kubernetes API calls needed to be made by the deployment pipeline.
- Fix issue where a few dozen concurrent deployment processes could exhaust maximum allowed connections originating from the Deployer service.
- Set scalability-minded options for resources deployed onto Kubernetes, significantly reducing CPU, memory, and network load at scale.
- Exposed configuration fields for many internal, as well as Kubernetes-facing, timeouts options.
- Fixed configuration issue that would cause Ambassador pods to be put up for eviction after only hundreds of models were deployed.

Documentation

- Added new page on node affinity and toleration.

- Added new page on Shapley values support.
- Added information on new Kubernetes options.
- Revised section on deploying models.

Version 0.52.1 (November 17, 2021)

New features

- Added support for Driverless AI (DAI) 1.10.0 (Supervised Models).
- Added new configuration options that let you push scoring data to a Kafka topic for monitoring purposes.

Improvements

- Experiments with metadata larger than 100 MB are now supported. The new limit is 1000 MB.

Version 0.52.0 (September 13, 2021)

New features

- Added support for Driverless AI (DAI) 1.9.3 Python pipelines.
- DAI Python pipelines must be imported either through the MLOps UI or programmatically by using the MLOps API to deploy. They cannot currently be deployed directly from the project.
- Ambassador timeout can now be configured per runtime with the `request-timeout` parameter in the Deployer configuration. Note that this parameter can also be set for any new BYOM runtime added to MLOps.

Improvements

- Added the ability to configure whether BYOM runtimes have write access to the volume hosting the model it's scoring.
- Exposed Terraform variables to make specifying custom BYOM entities easier.
- Added support for blob storages from public cloud storage services.
- Limited the number of error notifications displayed in the UI so that only one error is displayed at a time. Error notifications are now automatically cleared when the error condition disappears.

Version 0.51.0 (August 20, 2021)

Improvements:

- Implemented integration with Kafka for pushing scoring data.

Version 0.50.1 (August 04, 2021)

Improvements:

- Updated default Python runtimes with improved error handling.
- For secure environments, added a `terraform` flag for disabling BYOM.

Bug Fixes:

- For Python models, fixed a UI issue that caused complex deployments to be unsupported.

Version 0.50.0 (July 29, 2021)

New Features:

- Added support for third-party Python models.
    - Currently tested and supported versions include scikit-learn 0.24.2, PyTorch 1.9.0, XGBoost 1.4.2, LightGBM 3.2.1 and TensorFlow 2.5.0.
    - Added selectable artifact types and runtimes for all types of artifacts and models.

Improvements:

- Added new deployer endpoints for creating, listing and deleting deployments.

- Changed the MLOps client package name from `mlops` to `h2o_mlops_client`.
- Renamed deployment template input variable from `model_ingestion_image` to `model_ingest_image` to be consistent with the image name.
- Renamed deployment template input variable from `gateway_image` to `grpc_gateway_image` to be consistent with the image name.

Version 0.41.2 (June 2021)

Improvements:

- Added support for Driverless AI 1.9.3 MOJOs.

Version 0.41.1 (June 2021)

Improvements:

- Improved deployer logging.

Bug Fixes:

- Fixed an issue that caused installation through Terraform to not provide MLOps with all required configuration.

Version 0.41.0 (May 25, 2021)

Improvements:

- Added drag-and-drop option for importing Driverless AI MOJOs.

Bug Fixes:

- Fixed an issue that caused a broken download link to be generated for the MLOps gRPC-Gateway image.

Documentation:

- Added info on Driverless AI version compatibility.
- Added info on the MLOps API URL.
- Added info on the Token Endpoint URL.

Version 0.40.1 (March 15, 2021)

Improvements:

- Added alert messages to Grafana.
- Added pagination support for Project and DeployEnvironment list retrievals.
- Improved Model Fetcher logging.

Bug Fixes:

- Fixed an issue where some Model Fetcher processes were not checked for errors.
- Fixed an issue where some deployments got stuck in the Init phase when too many deployments started or restarted at the same time.
- Fixed a UI inconsistency between the Deployments and Models sections when no entries were displayed.
- Fixed a UI issue that caused the `Add new project` window to remain on the screen after successfully creating a project.
- Fixed an issue that allowed users to be registered without a username.
- Fixed an issue that caused models with one or more typos in their metadata to fail when deploying.
- Fixed an issue where H2O-3 models could not be deployed.
- Fixed an issue where some Driverless AI 1.9.1 models could not be deployed.

Version 0.40.0 (January 14, 2021)

New Features:

- Added Python API support.

Improvements:

- Added Model Fetcher to deploy scorers without a persistent volume.

Bug Fixes:

- Fixed an issue where the deployer remained in the 'Preparing' state indefinitely when a model had an unsupported transformer.
- Fixed an issue where models appeared in projects that they did not belong to.
- Fixed an issue that caused model selection to persist between different projects.
- Fixed an issue where the deployer did not clean up after fetching artifacts.
- Fixed an issue where certain menu items on the Projects page did not work as intended.

Version 0.31.3 (December 02, 2020)

Improvements:

- Driverless AI instances can now be run in a different namespace from storage namespace.
- Users can now override the default ingress class.

Bug Fixes:

- Fixed an issue that stopped project summary alerts from being updated.

Version 0.31.2 (November 11, 2020)

Improvements:

- Removed one PROD model per project restriction.
- Added a demo mode to the Studio page so that the default is more secure.
- Added optional password protection for Grafana.

Bug Fixes:

- Fixed an issue that stopped project summary alerts from being updated correctly.
- Fixed an issue that caused the alerts page to crash when a deployment had multiple alerts of mixed types.
- Fixed an issue that stopped the number of model pages from being updated correctly.
- Fixed an issue that caused all metadata to be fetched when listing experiments.

Version 0.31.1

Skipped and rolled in to 0.31.2.

Version 0.31.0 (October 21, 2020)

New Features:

- Added model endpoint security. Users can enable and configure authentication when deploying a model.

Bug Fixes:

- Fixed an issue where the sample cURL request for an endpoint with a hashed passphrase did not have an input box.
- Fixed an issue where single character passphrases were ignored.
- Fixed an issue where the set passphrase dialog did not appear for Champion/Challenger and A/B deployments.

Version 0.30.1 (October 08, 2020)

New Features:

- Added user-friendly H2O-3 model import support.

Improvements:

- Improvements in sorting/pagination.

Bug Fixes:

- Fixed issues with deployments list.
- Set default page size for lists to 10 pages.
- Various bug fixes

Version 0.30.0

Skipped and rolled in to 0.31.1.

Version 0.22.0 (July 30, 2020)

Bug Fixes:

- Fixed an issue where UI elements overlapped in Firefox.
- Fixed an issue where users could not log back in to MLOps after the session cookie expired.
- Fixed an issue where the Ambassador pod failed to start.

Version 0.21.1 (July 07, 2020)

Bug Fixes:

- Made the software version number visible in the UI.
- Added table pagination according to deployment pipeline design.
- Fixed an issue that caused the model actions drop-down menu to appear empty.
- Fixed an issue where models linked from Driverless AI could not be deleted.
- Fixed an issue where unfinished Driverless AI experiments could not be linked.
- Made the delete action unavailable to users with the Reader role.
- Fixed an issue where deployments were reported as having failed after pods were restarted.
- Fixed an issue where the scoring data for an experiment linked to more than one project was not stored in InfluxDB.

Version 0.21.0 (June 12, 2020)

New Features:

- Added drift detection analysis for models.
- Added A/B testing to compare the performance of two or more models.
- Added Champion/Challenger deployments.

Bug Fixes:

- Increased the default timeout for waiting for a pod to provision when deploying.
- Fixed an issue that stopped deployments from being listed for challenger models.
- Fixed an issue that caused MLOps to crash when a feature field was not found in the drift report.
- Fixed an issue that caused the A/B Test link to remain active when no model was selected.
- Fixed an issue on the Projects page that caused the delete model action to not work correctly.
- Fixed an issue in the Grafana dashboard that caused the scoring latency graph to appear as having no data.
- Fixed an issue that stopped collaborators from being able to create deployments when they were not restricted from doing so.
- For the Reader user role, fixed an issue that caused incomplete error messages to appear for failed user actions.
- Fixed an issue that caused the filtering option to disappear from the Models page.
- Fixed an issue where undeploying a model that was a part of multiple deployments did not work correctly.
- Fixed an issue that caused the 'More details' action to become activated when 'Monitoring' was selected from the Actions menu.

Version 0.20.1 (April 02, 2020)

Bug Fixes:

- Fixed an issue that stopped the user interface from accessing storage after restarting all pods.
- Fixed an issue that caused PostgreSQL data to be purged when the pod was restarted.

Version 0.20.0 (April 01, 2020)

- First stable release.

Migration guide

This guide helps you update H2O MLOps when moving between versions. It outlines migration steps for each version upgrade and focuses on changes that require updates to your code, configuration, or workflows.

For detailed changes to the H2O MLOps Python client, see the Python client migration guide

From 0.70.0 to 1.0.0

Workspace integration

MLOps 1.0.0 is integrated with the Workspace service. All projects have been migrated to Workspaces, and both the user interface and Python client have been updated accordingly.

Python client

The legacy, automatically generated Python Client is no longer compatible with MLOPS 1.0.0. Only the h2o-mlops 1.4.0 and higher is supported. Please migrate your workflows to the new Python Client.

To migrate from version 1.3.x to 1.4.x, see the Python client migration guide from v1.3.x to v1.4.x.

Removal of Wave UI

Starting from H2O MLOps version 1.0.0, the legacy Wave-based user interface is no longer available. The official and supported MLOPs user interface is now part of the H2O AI Cloud user interface. The Admin Analytics Wave app has also been removed, and its capabilities have been migrated to the new interface.

Helm chart changes

In `Affinity` and `Tolerations` configuration, the field `matchExpression` has been replaced by the `matchExpressions`.

The option `apiGateway.authorization.enabled` has been removed as authorization is now automatically used on API Gateway. In case this option was set to `false` in previous MLOPs versions, please make sure to keep the `apiGateway.authorization.allowedUserRoles` set to `[]`. `[]` is default value of this option.

Monitoring setup changes

Starting from H2O MLOps version 1.0.0, the legacy monitoring setup has been removed and replaced with a new configuration method using the `MonitoringOptions` class in the Python client. Monitoring is disabled by default and must be explicitly enabled during or after deployment.

Kafka sink configuration is now more granular and can be set on a per-model basis.

For more information, see Monitoring setup.

Migrate from old monitoring to new monitoring New monitoring setup is completely different from legacy monitoring setup and if you need to migrate legacy monitoring data and deployments to new monitoring setup MLOps provides a migration job with MLOps 1.0.0 release. You need to explicitly enable it during or after the installation using Helm charts parameters.

Monitoring migration is anyway optional and new monitoring will work without it for newly created deployments. But new monitoring won't be enabled for already created deployments before MLOps 1.0.0 and you won't see monitoring metrics for historical scoring data without the migration. If you need to enable the monitoring migration you have to set the helm parameters as follows.

```
global:
  components:
    influxdb:
      enabled: true
    superset:
      enabled: true
mlops:
  config:
    models:
```

```
monitoringEnabled: true
monitoringMigrationEnabled: true
```

Please note that to enable monitoring migration you need to enable new monitoring also. You cannot enable only the migration alone.

Hash security option changes

Starting from H2O MLOps version 1.0.0, hash-based security options require you to provide the passphrase directly. The hashing is now handled automatically in the backend.

Make sure to store the passphrase in a secure location, as you won't be able to retrieve it after it's submitted.

From 0.69.x to 0.70.0

Transition from Scoring Client to native batch scoring

Starting from H2O MLOps version 0.70, batch scoring functionality has been natively integrated into H2O MLOps, replacing the H2O MLOps Scoring Client. The native batch scoring implementation is available through the official H2O MLOps Python client.

For added convenience, batch scoring can also be performed through the new H2O MLOps UI. Please contact H2O.ai support in case you need any guidance on this migration.

Workload identity and IAM authentication

Starting from H2O MLOps version 0.70, workload identity and IAM authentication will be managed using the `github.com/h2oai/go-pkg/database/postgres/v2` library for the `mlops-storage`, `mlops-telemetry`, and `mlops-deployer` components.

Update the connection strings for these components to match the formats shown in the examples below:

Example of the `mlops-storage` and `mlops-telemetry` database connection string:

```
storage_db_connection_string = "postgres://${var.mlops_db_username}@${var.mlops_db_address}:5432/${var.mlops_
```

Example of the `mlops-deployer` database connection string:

```
deployment_db_connection_string = "postgres://${var.mlops_deployment_db_address}/${var.mlops_deployment_db_na
```

Removal of mTLS

mTLS is no longer managed by Kubernetes jobs. For environments requiring mTLS communication between MLOps services, this should now be handled by a service mesh solution (such as Istio).

Previous versions used SPIFFE for service-to-service authentication. Version 0.70+ now uses service account tokens instead.

Migration Steps If your deployment requires mTLS:

1. Remove any existing Kubernetes job configurations for mTLS.
2. Implement a service mesh solution to manage mTLS between services.
3. Configure your service mesh to handle the TLS certificate management.

Changes in Helm

- Config `tls` has been removed.
- Config `storage.tls` has been removed.
- Config `deployer.tls` has been removed.
- Config `ingest.tls` has been removed.
- Config `apiGateway.tls` has been removed.
- Config `monitoringAppBackend.tls` has been removed.

- Config `deployer.telemetry.auth.tlsEnabled` has been removed.
- Config `telemetry.serverSecurityEnabled` has been removed.
- Config `storage.auth.service` has been added:

```
service:
  # -- Issuer URL for service authentication.
  # In general, this should be set to whatever the "issuer" field is in the cluster's
  # OIDC discovery document.
  # `kubectl get --raw /.well-known/openid-configuration` can be used to retrieve
  # the discovery document.
  issuerURL: "https://kubernetes.default.svc"

  # -- Configures whether to validate issuer URL for service authentication.
  # Issuer of some service account variants does not need to be the issuer
  # specified by the issuerURL.
  validateIssuer: false

  # -- Configures whether to use the Kubernetes HTTP client with TLS and token the issuer discovery
  # and downloading the signing keys.
  # Disable this when the issuer is not Kubernetes API server.
  useKubernetesHTTPClient: true
```

Removal of support for older H2O Driverless AI versions

In MLOps version 0.70, support for H2O Driverless AI versions 1.10.6.3 and earlier has been discontinued. This change affects the following versions:

- 1.10.5-cuda11.2.2
- 1.10.5.1-cuda11.2.2
- 1.10.6-cuda11.2.2
- 1.10.6.1-cuda11.2.2
- 1.10.6.2-cuda11.2.2
- 1.10.6.3-cuda11.2.2

Removal of `Pickle` Runtime

Starting from H2O MLOps version 0.70, the `Pickle` Runtime has been removed. ### Removal of `environment` from Python Client and UI

Starting with MLOps version 0.70.0, the `environment` feature has been removed from the user perspective in both the Python client and the UI. This change does not apply to the backend, and environment-related functionalities remain intact.

Changes in the UI

- Users no longer need to select an environment (e.g., PROD or DEV) when creating a deployment.
- The environment now defaults internally to `PROD`.
- Environment-related details are no longer visible in the UI.

Changes in the Python Client

- The `environments` property of the `MLOpsProject` instance is no longer available starting from client version 1.3.0.
- The environment now defaults internally to `PROD`.
- When using the updated client, the following adjustments must be made in the code. Here, `project` refers to an instance of `MLOpsProject`, and `client` refers to an instance of `h2o_mlops.Client`:

  Code Adjustments:

  - Replace `project.environments.get(uid).deployments` with `project.deployments`.
  - Replace `project.environments.get(uid).endpoints` with `project.endpoints`.
  - Replace `project.environments.get(uid).allowed_affinities` with `client.allowed_affinities`.

- Replace `project.environments.get(uid).allowed_tolerations` with `client.allowed_tolerations`.

With these adjustments, your code will remain compatible with both the updated and older versions of the MLOps backend.

From 0.68.x to 0.69.0

MLOPs runtimes

All runtime images must be updated to at least version 1.5.3, which was released with H2O MLOps version 0.69.0

Starting with version 0.69.0 all runtime images must be always updated to the runtime images released with that corresponding H2O MLOPs version. For example, H2O MLOps 0.69.1 was released with runtime images v1.5.4, and therefore all deployment's images must be updated to runtimes v1.5.4.

MLOps storage

Starting with MLOps version 0.69.0, only blob storages are supported as the backend. Support for other storage options has been discontinued. This change impacts the configuration parameters used in the MLOps Helm charts.

Changes to storage configuration parameters

```
storage:
  persistence:
    # All parameters under this section are no longer supported.
  cloudPersistence:
    # The 'enabled' parameter was removed since cloudPersistence is now the only option supported.
    enabled:
  pvcMigration:
    # All parameters under this section are no longer supported.
```

Note: After upgrading to MLOps 0.69.0, you can safely delete your existing PVC that was used as the storage backend prior to the 0.68.0 release. Perform this step manually to prevent any unintended data loss.

PBKDF2 hash support

H2O MLOps v0.69.0 now supports the **PBKDF2** passphrase hash algorithm for more secure hashing. Note the following details:

- The **PBKDF2** hash should follow the format `pbkdf2:<hashFunc>:<iterations>$<salt>$<hash>`.
- The `salt` and `hash` should be base64 encoded.
- **PBKDF2** hashing replaces **bcrypt** when creating deployments with the Passphrase (Stored hashed) security option.
- The Passphrase (Stored hashed) security option is listed as an available option in the **Create Deployment** panel dropdown only if `PASSPHRASE_HASH_TYPE_PBKDF2` is included under `securityOptions.activated` in the `values.yaml`. Having `PASSPHRASE_HASH_TYPE_BCRYPT` is neither sufficient nor required.
- Older deployments created with **bcrypt** hashing remains accessible without requiring any additional configuration.

From 0.67.x to 0.68.0

(Optional) Vertical Pod Autoscaler (VPA) support

MLOps version 0.68.0 introduces Vertical Pod Autoscaler (VPA) support for the Deployer. Note that VPA activation is optional and performed upon request. VPA allows dynamic scaling of CPU and memory resources based on application usage, improving resource efficiency and optimizing costs.

If the VPA is activated in MLOps, then VPA is supported in the cluster and the VPA CRDs and controllers are up and running alongside the Metrics Server.

For more information, see the Installation section of the VPA GitHub README and the Metrics Server installation instructions.

Note: For a list of known limitations, see the Known limitations section of the VPA GitHub README.

Key changes

- VPA Resource Specifications: Added VPA resource specification logic to the Scoring Apps and App Composer, allowing for the dynamic adjustment of their resource limits based on real-time demand.
- API Updates: New API logic has been added for specifying and validating VPA resources.
- New VPA Utility Functions: Implemented utility methods for creating and managing VPA resources, including validation and resource quantity handling.
- Deprecated Function Removal: Removed the deprecated Fabric8 createOrReplace usage in the Scoring Apps.

Removal of HT runtime based on Python 3.8

The Hydrogen Torch (HT) runtime based on Python 3.8, which was available by default in MLOps version 0.67.x, has been removed as of MLOps version 0.68.0. However, you can still use this runtime by registering it through extra runtimes.

The following requirements need to be met so that the runtime registered through extra runtimes is also visible in the UI:

- The `mlflow/flavors/python_function/loader_module` must match `mlflow.pyfunc.model`.
- The runtime name must adhere to this pattern: `(python-scorer_hydrogen_torch_)(\w*)(38)(\w*)`.

Configure maximum number of Kubernetes replicas

With H2O MLOps v0.68.0, you can configure the maximum number of Kubernetes replicas that can be specified when creating a new deployment. To do this, update `maxDeploymentReplicas` in the `values.yaml` file (`charts/mlops/values.yaml`). By default, the `maxDeploymentReplicas` value is set to 5.

Removal of MLflow runtimes based on Python 3.8

MLflow runtimes based on Python 3.8 have been removed in MLOps version 0.68.0. Python 3.8 has officially reached end of life as of October 07, 2024.

Pickle runtime based on Python 3.12

MLOps version 0.68.0 introduces a pickle runtime using Python 3.12. Choose one of the following options:

- Update your models to work with Python 3.12.
- If you cannot update your models, the original pickle runtime based on Python 3.8.18 can be configured during MLOps installation by replacing the `pickle-3.12.7` image with `pickle-3.8.18`.

Deployment of MLOps Telemetry as a long-running microservice

In MLOps version 0.67 and earlier, the MLOps `telemetry` component was configured as a cron job within the MLOps `storage` component in the Helm configuration. Starting with MLOps version 0.68, the MLOps `telemetry` component must be deployed as a separate long-running microservice that publishes event data at scheduled intervals.

To migrate from MLOps version 0.67 to 0.68: 1. Remove the cron job configuration from the MLOps `storage` component in the Helm configuration. 2. Implement it as a separate `telemetry` component within Helm.

Helm values must be set as follows:
```
# Telemetry Configrations   telemetry = {     enabled = true     image
= {       repository = "h2oai-modelstorage-telemetry${local.shared_services_repository_suffix}"
tag        = local.component_version.mlops_telemetry_version     }     replicaCount = 1
nodeSelector = {       "hac.h2o.ai/provisioner" = "karpenter"     }     tolerations = [
{         key      = "type"         operator = "Equal"         value    = "cpu-consolidation"
effect   = "NoSchedule"       }     ]     podSecurityContext = {       enabled = true     }
containerSecurityContext = {       enabled = true     }     serviceAccount = {       name
= "hac-mlops-storage-telemetry-service-account"     }     serverAddress = "hac-telemetry-
service.telemetry.svc.cluster.local:80"     config = {       logLevel = "error"     }   }
```

Scheduler routine for MLOps Telemetry

MLOps version 0.68.0 introduces the `SCHEDULER_INTERVAL_SECONDS` env variable to run scheduler routine inside the application itself, replacing the use of a cron job. As a result, MLOps Telemetry is deployed as a long-running deployment in the K8s cluster that publishes event data at scheduled intervals. The default value is as follows:

```
SCHEDULER_INTERVAL_SECONDS=300
```

Restructured environment security options

Environment-related security options are now configured in a different way. Prior to v0.68.0, security options were specified using their corresponding numerical values. For example:

```
securityOptions: [1,2,3]
```

From v0.68.0 onwards, activated security options are configured in the `values.yaml` file (`charts/mlops/values.yaml`) using the security option name. For example:

```
securityOptions:
    activated:

        - .......
        - "AUTHORIZATION_PROTOCOL_OIDC"
        - .......
```

You can also set the default security option in the `values.yaml` file (`charts/mlops/values.yaml`) using the security option name. The default option serves as the default security setting that will be applied in the UI when creating a deployment and it must be a part of the Activated Security Options List.

```
securityOptions:
    activated:

        - .......
        - "PASSPHRASE_HASH_TYPE_PLAINTEXT"
        - .......
    default: "PASSPHRASE_HASH_TYPE_PLAINTEXT"
```

The following security options are supported in v0.68.0:

- **DISABLED**: No security options are activated.
- **PASSPHRASE_HASH_TYPE_PLAINTEXT**: Passphrase hash type is plaintext.
- **PASSPHRASE_HASH_TYPE_BCRYPT**: Passphrase hash type is bcrypt.
- **AUTHORIZATION_PROTOCOL_OIDC**: OIDC authorization protocol is activated.

Notes:

- The Activated Security Options List can not be empty.
- The default option must be part of the Activated Security Options List.

From v0.68.0 onwards, the way to create a deployment with No Security via API call also differs from previous versions. This change includes the following modifications to the `h2o-mlops` Python Client:

- `security_options` is now a required field for the `create_single` method of the `MLOpsScoringDeployments` class.
- To ensure backward compatibility, v0.68.0 includes a new attribute for the `SecurityOptions` class, called `disabled_security`. This attribute allows handling cases with the No Security option by setting it to True, instead of treating None or SecurityOptions() as No Security.
- Users of MLOps assembly v0.68.0 or above must set `disabled_security=True` to use the No Security option. For users on older versions, No Security mode can be accessed by using SecurityOptions with default values.

Helm changes

- As of version 0.68.0, the `ENABLE_USER_EXTERNALID_UPDATE` environment variable has been removed from storage, as it is no longer necessary.
- `deploymentEnvironment.corsOrigin` has been removed. Use `global.cors.allowedOrigin` instead.

Default deployment security option

As of version 0.68.0, the default security option for deployment is `PASSPHRASE_HASH_TYPE_PLAINTEXT`. Prior to this version, deployments were not secured by default.

Cloud migration information: MLOps storage

Starting with version 0.68.0, H2O MLOps will no longer support PVCs for storage, transitioning instead to cloud blob storage. MLOps storage will support blob storage from all three major cloud providers—AWS, Azure, and GCP—as well as Minio for on-premises installations. Consequently, all existing data must be migrated from PVC to blob storage during the upgrade to MLOps 0.68.0. All the data migrations steps will be taken care of by MLOps when MLOps storage is deployed in the MIGRATE mode and no manual user intervention is needed. End users shouldn't experience any down time or data loss while the migration is in progress.

Installation instructions

Deploy storage in MIGRATE mode Note: Only follow the instructions in this section if MLOps storage was previously deployed with LOCAL mode using a Kubernetes PVC as the storage.

For AWS environments with S3

IAM auth is used to access the bucket. Following annotation should be set to the storage service account.

```
eks.amazonaws.com/role-arn: <iam-role-arn>
```

```
storage:
  serviceAccount:
    create: true
    annotations: {
      eks.amazonaws.com/role-arn: <iam-role-arn>
    }
  persistence:
    enabled: true
  cloudPersistence:
    enabled: true
    url: s3://<bucket-name>?region=<bucket-region>&prefix=<optional-prefix>
  pvcMigration:
    enabled: true
    cloudProvider: s3
    bucketName: <bucket-name>
    region: <bucket-region>
    prefix: <optional-prefix>
```

For GCP environments with Google Cloud Storage

Workload identify is used to access the bucket. The following annotation must be set to the storage service account:

```
iam.gke.io/gcp-service-account: <service_account_email>
```

Helm values must be set as follows:

```
storage:
  serviceAccount:
    create: true
    annotations: {
      iam.gke.io/gcp-service-account: <service_account_email>
    }
  persistence:
    enabled: true
  cloudPersistence:
    enabled: true
    url: gs://<bucket-name>
  pvcMigration:
    enabled: true
    cloudProvider: gcs
    bucketName: <bucket-name>
    region: <bucket-region>
```


For Azure environments with Azure Blob Storage

Workload identify is used to access the bucket. The following annotation must be set to the storage service account:

```
azure.workload.identity/client-id=<client-id>
```

The following label must be set to storage pods (service and migrator job):

```
azure.workload.identity/use=true
```

Helm values must be set as follows:

```
storage:
  serviceAccount:
    create: true
    annotations: {
      azure.workload.identity/client-id=<client-id>
    }
  extraPodLabels: {
    azure.workload.identity/use=true
  }
  persistence:
    enabled: true
  cloudPersistence:
    enabled: true
    url: azblob://<bucket-name>
  pvcMigration:
    enabled: true
    cloudProvider: azureblob
    bucketName: <bucket-name>
    region: <bucket-region>
    accountName: <storage-account-name>
```

For on-premise environments with Minio

```
storage:
  persistence:
    enabled: true
  cloudPersistence:
    enabled: true
    url: s3://<minio-bucket-name>?endpoint=<minio-url>&region=<minio-region>&hostname_immutable=true
    access_key_id: <minio-access-key-id>
    secret_access_key: <minio-secret-access-key>
  pvcMigration:
    enabled: true
    cloudProvider: minio
    bucketName: <bucket-name>
    region: <minio-region>
    endpoint: <minio-url>
    access_key_id: <minio-access-key-id>
    secret_access_key: <minio-secret-access-key>
```

From 0.66.1 to 0.67.0

Announcement: Upcoming Java MOJO Runtime removal

The Java MOJO Runtime will be removed in the 0.69.0 MLOps release. Version 0.68.0 will be the last release to include the Java MOJO Runtime.

Users are advised to migrate to the C++ MOJO Runtime, which is a 1:1 mapping of the Java runtime that accepts a wider range of algorithms Driverless AI may use that the Java runtime does not support, including BERT, GrowNet, and TensorFlow models.

Scoring runtimes

- MLflow Runtimes images are twice as large now. This means that deployments of these run-times can take longer due to longer pulling times.
- Runtimes for DAI 1.10.4.3 and older are removed as of MLOps version 0.67.0.
- MLflow runtimes support Python 3.8 and later starting with MLOps version 0.67.0.

For more information on scoring runtimes in H2O MLOps, see Scoring runtimes.

Python client

Starting with version 0.67.0, the official Python client of H2O MLOps is `h2o-mlops`. The minimum Python version required for the client is Python 3.9.

Built on top of the legacy Python client, `h2o-mlops` retains all previous functionalities. You can continue to access the legacy client's features through `h2o-mlops` as needed.

Note that users of the legacy client can switch to the new Python client (`h2o-mlops`) by importing `h2o-mlops` before using any features of the legacy client. This switch can be made without needing to modify any existing code or import statements.

Removal of Conda from Wave app

With the removal of Conda as of MLOps version 0.67.0, third-party models can no longer be uploaded to the MLOps frontend using serialized Pickle files. However, you can still upload models from frameworks like scikit-learn, PyTorch, XGBoost, LightGBM, and TensorFlow using MLflow packaged files.

Monitoring data retention

- Starting with version 0.67.0, per project data retention duration can be set for monitoring data stored on InfluxDB. To enable this feature, set the `MONITOR_INFLUXDB_PER_PROJECT_DATA_RETENTION_PERIOD` env to the deployer with a correct duration string. Minimum retention period is 1h and the max is `INF`. `INF` will be the default If `MONITOR_INFLUXDB_PER_PROJECT_DATA_RETENTION_PERIOD` is not set, `INF` is the default duration.

  `-monitor_influxdb_per_project_data_retention_period` is exposed for H2O MLOps helm charts to set the `MONITOR_INFLUXDB_PER_PROJECT_DATA_RETENTION_PERIOD` for deployer.

Emissary

Switch from emissary to gateway-api:

- Emissary's CRDs are no longer used.
- For mapping deployments to http, Gateway API's HTTPRoute CRD is used.
- Gateway API implemented with Envoy Gateway.
- (**Breaking change**) Gateway API doesn't support custom error responses. This means that if a deployment is scaled down, the following custom error body is no longer displayed: `Deployment is scaled down to zero replicas. Please increase the number of replicas to use the deployment.` For more information, see Custom error responses.
- (**Breaking change**) - If a deployment is scaled down, error code **500** is thrown instead of **503**.

Other changes

- External model registry is removed as of version 0.67.0.

Key terms

This page provides an overview of key terms and concepts that apply to H2O MLOps.

Workspaces

In MLOps, a workspace is the main folder that contains experiments, artifacts, models, and deployments. Workspaces are designed to be collaborative, and can be shared between multiple individuals. Additionally, workspace owners can specify role-based access control for each individual that is invited to collaborate on a workspace. Workspaces can be used to group all work items for a specific team, or can be used to group all work items for a specific use case.

:::info note - Access to users is controlled at the workspace level. If a user has read and write access to a workspace, they are able to make changes to all experiments, models, and deployments associated with that workspace. - Any projects that have been created in H2O Driverless AI are automatically synchronized with H2O MLOps workspaces. :::

BYOM (Bring Your Own Model)

BYOM, or Bring Your Own Model, refers to the process of importing models trained outside the H2O MLOps platform, such as models from H2O Driverless AI, H2O-3, or MLflow, for deployment and management in H2O MLOps.

Experiments

In MLOps, an experiment is defined as the output of a training job. Many different experiments can be rapidly created by modifying specific parameters and hyperparameters. Experiments can be imported in the following formats:

- Driverless AI MOJO (directly through DAI interface, or by dragging and dropping file).
- H2O-3 open source MOJO (dragging and dropping file).
- Third-party model frameworks. This includes scikit-learn, PyTorch, XGBoost, LightGBM, and TensorFlow. Import by dragging and dropping an MLflow packaged file.

:::info note Before an experiment can be deployed, it must first be registered in the H2O MLOps Model Registry. :::

Experiment metadata

Each experiment in the H2O.ai Storage can have multiple key-value pairs attached to it. These values are not interpreted by the storage itself but can be interpreted by the clients or client services that access data in the Storage.

Experiments that are added to H2O MLOps from the MLflow Model Registry include both the MLflow model name (`source_model_name`) and MLflow version number (`source_model_version`) as part of the experiment metadata.

Artifacts in MLOps

Defining artifacts and experiment artifacts

- **Artifact**: An arbitrary binary large object (BLOB) attached to a particular entity in the H2O.ai Storage.
- **Experiment Artifact**: Any artifact that is attached to the experiment entity.

Artifact type

Because any entity can have multiple artifacts attached to it, specific artifacts must be identified by their type. Type is an arbitrary string. Artifact type is recognized by and relevant to MLOps deployments.

The following is a list of artifact types:

- `dai/mojo_pipeline` (Natively supported, ingestion supported)
- `h2o3/mojo` (Natively supported, ingestion supported)
- `Python/MLflow` (Ingestion supported)

Deployable artifact type

A deployable artifact type is an artifact type that the Deployer knows how to process and deploy. Each deployable artifact type consists of the name, readable name, and reference to the artifact type.

Artifact processor

Artifact processor is the routine that takes the raw artifact data and transforms it into the format that is digestible by the runtime. In this routine, an artifact is defined by its name, the model type that it produces, and a container image reference.

Artifact processors can be any container image that can be pulled from the target deployment environment. Each processor needs to recognize and use two environmental variables.

The following is a list of artifact processor environment variables:

- `SOURCE_PATH`: Path to the file containing the raw data of the artifact
- `TARGET_PATH`: Path where the processor saves its output. This path is passed as `MODEL_PATH` to the runtime

Deployments

In MLOps, deployments are created when model version(s) are served for scoring. Deployments are configured by Type (real-time, batch) and Mode (single model, A/B, C/C).

A deployment is always tied to a specific model version. You cannot edit or replace the version of an existing deployment. To serve a new model version, you must create a new deployment.

To minimize disruption, endpoints can be reused across deployments by deploying the new model and updating the endpoint target from the old deployment to the new deployment. This allows you to continue using the same endpoint with zero downtime.

Drift detection

Drift detection in MLOps is based on **Feature Drift**. This term is used to describe situations where the input values for features during scoring differ from the input values for features during training. When drift increases, it means that the model is seeing data that it was not trained on, and so the performance and results of the model may not be accurate.

Drift evaluation

The drift evaluation metrics used in MLOps are the **Population Stability Index** (PSI) and **Drift Score**. The PSI only works for numerical features, whereas drift score can work with categorical features.

The following image compares PSI, drift score and AUC.

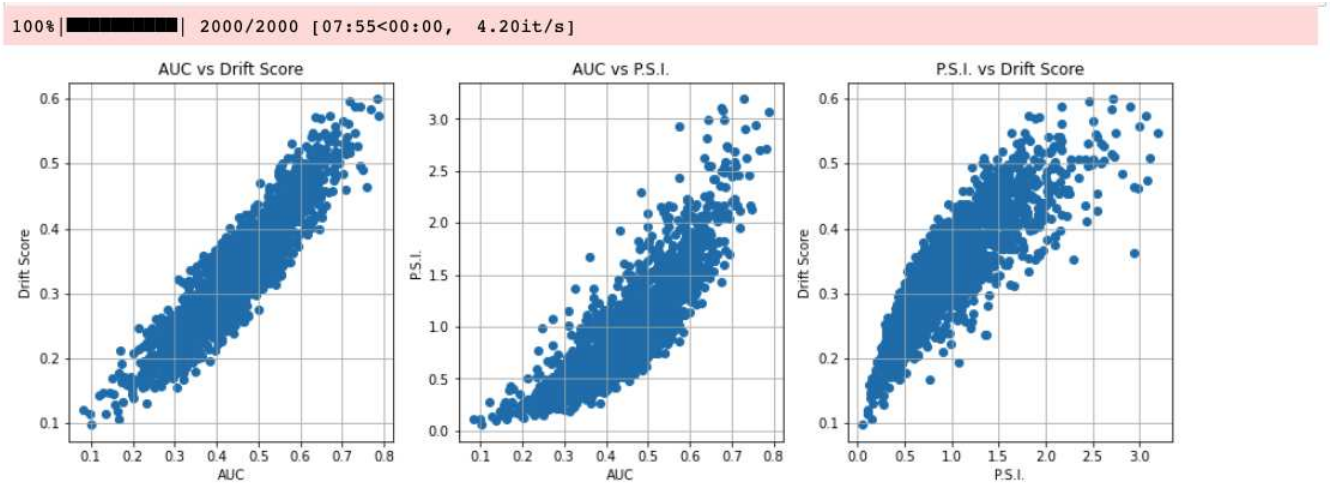


Figure 1: PSI, drift score, and AUC comparison

Population Stability Index To learn how the PSI is calculated in MLOps, refer to the following steps:

1. The reference samples are binned in a maximum of 10 equal bins. Depending on the distribution, you may end up with less or unequally populated bins. Equal binning gives less weight to tails.
2. Compute the frequency of each bin.
3. Apply the binning to scoring samples and compute frequencies.
4. Compute PSI as follows:

$$\sum_{b=1}^{10}(train_b - scoring_b) * ln\left(\frac{train_b}{scoring_b}\right)$$

Figure 2: PSI formula

:::info note - PSI does not support missing values. - PSI is more suited for numerical features or ordinal features. This metric may have difficulty with categorical features, particularly with high cardinality categoricals. :::

Drift score To learn how drift score is calculated in MLOps, refer to the following steps:

1. The reference samples are binned in a maximum of 10 equal bins. Depending on the distribution, you may end up with less or unequally populated bins. Equal binning gives less weight to tails.
2. Compute the frequency of each bin.
3. Apply the binning to scoring samples and compute frequencies.
4. Compute drift score as follows:

$$\frac{1}{2}\sum_{b=1}^{10} abs(train\_freq_b - scoring\_freq\_b)$$

Figure 3: Drift score formula

Node affinity and toleration

As stated in the official Kubernetes documentation, "node affinity is a property of Pods that attracts them to a set of nodes, either as a preference or a hard requirement. Taints are the opposite—they allow a node to repel a set of pods. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints." In the case of MLOps, these options let you ensure that scorers (pods) are scheduled onto specific machines (nodes) in a cluster that have been set up for machine learning tasks.