Cloudera Data Science Workbench

Models

Date published: 2020-02-28
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Models

This topic describes the challenges and solutions that models address.

Challenge

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

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

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

Solution

Starting with version 1.4, Cloudera Data Science Workbench allows data scientists to build and deploy their own models as REST APIs. Data scientists can now select a Python or R function within a project file, and Cloudera Data Science Workbench will:

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

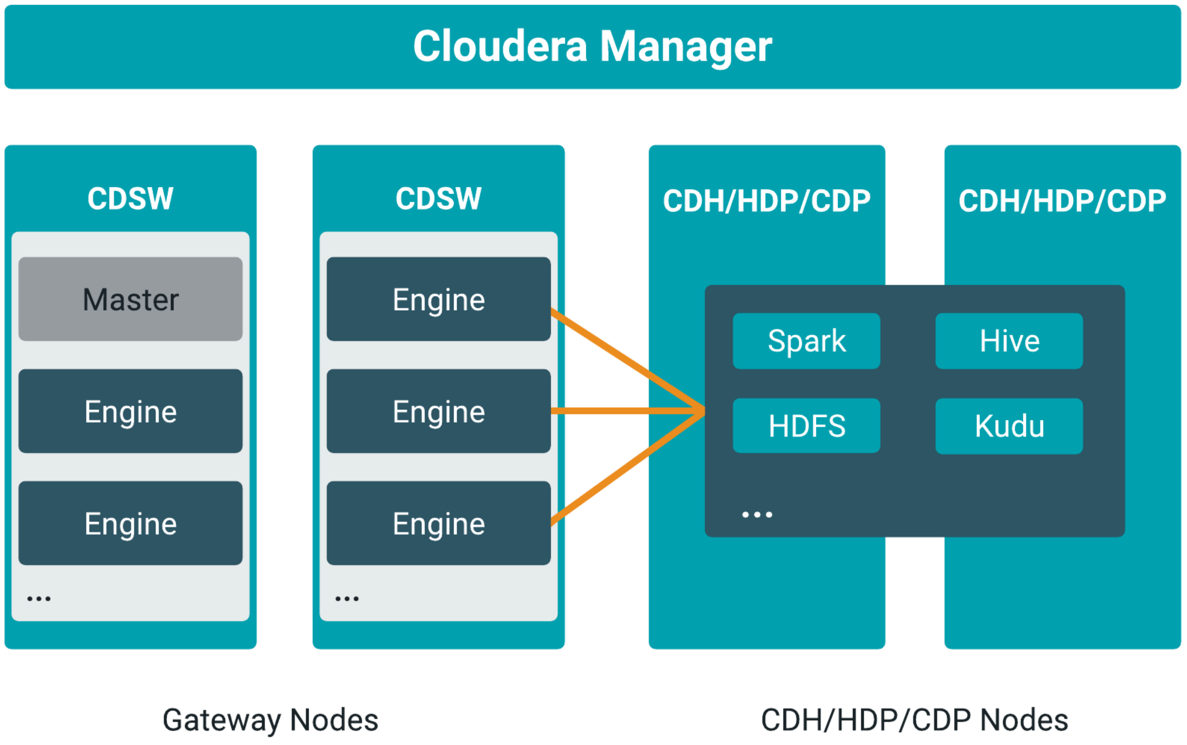
Introduction to Production Machine Learning

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

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

Core capabilities

- Seamless portability across private cloud, public cloud, and hybrid cloud powered by Kubernetes
- Fully containerized workloads - including Python, and R - for scale-out data engineering and machine learning with seamless distributed dependency management
- High-performance deep learning with distributed GPU scheduling and training
- Secure data access across HDFS, cloud object stores, and external databases



CDSW users

CDSW users are:

- Data management and data science executives at large enterprises who want to empower teams to develop and deploy machine learning at scale.
- Data scientist developers (use open source languages like Python, R, Scala) who want fast access to compute and corporate data, the ability to work collaboratively and share, and an agile path to production model deployment.
- IT architects and administrators who need a scalable platform to enable data scientists in the face of shifting cloud strategies while maintaining security, governance, and compliance. They can easily provision environments and enable resource scaling so they - and the teams they support - can spend less time on infrastructure and more time on innovation.

Challenges with model deployment and serving

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

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

Challenges with model monitoring

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

You can monitor the performance of the model on two levels: technical performance (latency, throughput, and so on similar to an Application Performance Management), and mathematical performance (is the model predicting correctly, is the model biased, and so on).

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

Time series metrics: Metrics measured in-line with model prediction. It can be useful to track the changes in these values over time. It is the finest granular data for the most recent measurement. To improve performance, older data is aggregated to reduce data records and storage.

Post-prediction metrics: Metrics that are calculated after prediction time, based on ground truth and/or batches (aggregates) of time series metrics. To collect metrics from the models, the Python SDK has been extended to include the following functions that you can use to store different types of metrics:

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

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

- Tracking accuracy of a model over time
- Tracking drift

Use case 1: Tracking accuracy of a model over time

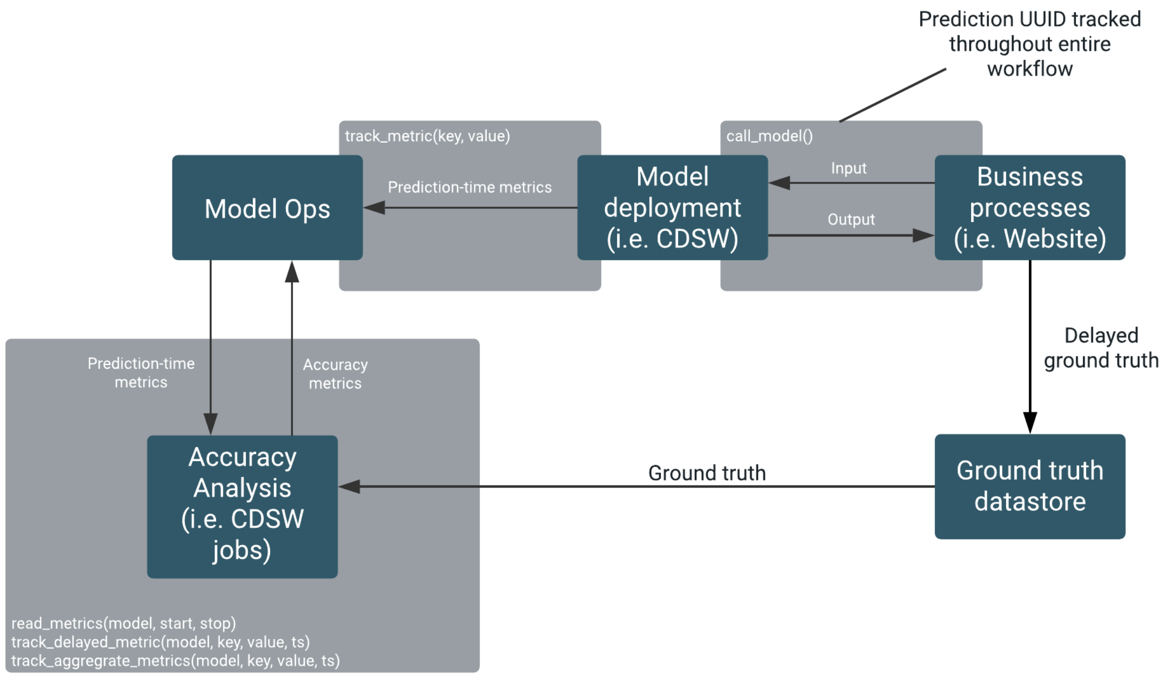
Consider the case of a large telco. When a customer service representative takes a call from a customer, a web application presents an estimate of the risk that the customer will churn. The service representative takes this risk into account when evaluating whether to offer promotions.

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

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

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

The following diagram illustrates use cases like this one:


Prediction UUID tracked throughout entire workflow
track_metric(key, value)
call_model()
Model Ops
Prediction-time metrics
Model deployment (i.e. CDSW)
Input
Output
Business processes (i.e. Website)
Delayed ground truth
Prediction-time metrics
Accuracy metrics
Accuracy Analysis (i.e. CDSW jobs)
Ground truth
Ground truth datastore
read_metrics(model, start, stop)
track_delayed_metric(model, key, value, ts)
track_aggregrate_metrics(model, key, value, ts)

The ground truth can be stored in an external datastore, such as Cloudera Data Warehouse or in the metrics store.

Use case 2: Tracking drift

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

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

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

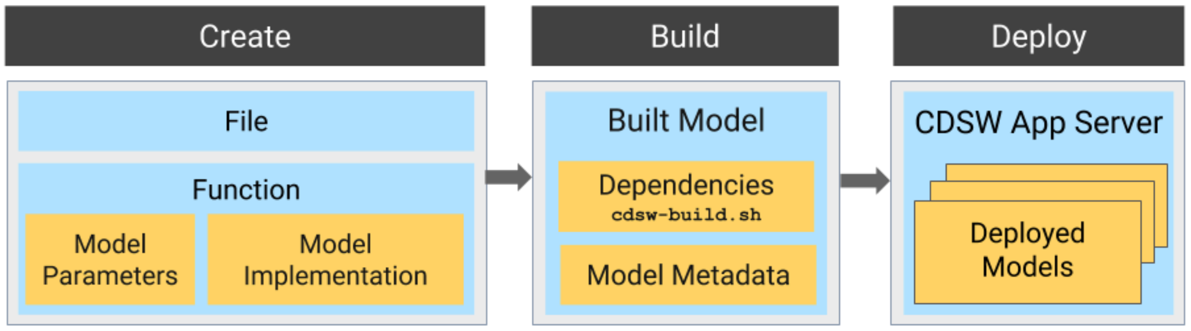
Concepts and Terminology

This topic provides model conceptual information.

Model

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

Stages of the Model Deployment Process



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

Create

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

  The function will likely include the following components:

  - Model Implementation

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

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

Build

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

- Built Model

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

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

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

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

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

Deploy

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

- Deployed Model

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

  You can set environmental variables each time you deploy a model. Note that models also inherit any environment variables set at the project and global level. However, in case of any conflicts, variables set per-model will take precedence.

  Note:

  - If you are using any model-specific environmental variables, these must be specified every time you re-deploy a model. Models do not inherit environmental variables from previous deployments.
  - Note that custom mounts or environment variables configured in cdsw.conf (such as NO_PROXY, HTTP(S)_PROXY, etc.) are still not passed to the container builds for experiments and models (even though they are applied to sessions, jobs, and deployed models/experiments).
- Model Replicas

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

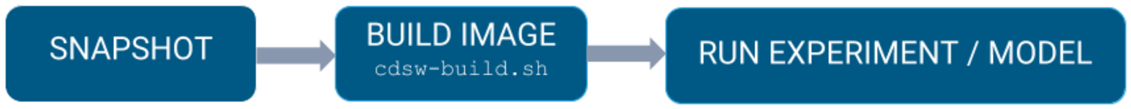
  Deployment IDs are numeric IDs used to track models deployed across Cloudera Data Science Workbench. They are not bound to a model or project.

Engines for Experiments and Models

In Cloudera Data Science Workbench, models, experiments, jobs, and sessions are all created and executed within the context of a project.

We've described the different ways in which you can customize a project's engine environment for sessions and jobs here. However, engines for models and experiments are completely isolated from the rest of the project.

Every time a model or experiment is kicked off, Cloudera Data Science Workbench creates a new isolated Docker image where the model or experiment is executed. This isolation in build and execution makes it possible for Cloudera Data Science Workbench to keep track of input and output artifacts for every experiment you run. In case of models, versioned builds give you a way to retain build history for models and a reliable way to rollback to an older version of a model if needed.



The rest of this topic describes the engine build process that occurs when you kick off a model or experiment.

Snapshot Code

When you first launch an experiment or model, Cloudera Data Science Workbench takes a Git snapshot of the project filesystem at that point in time. It is important to note that this Git server functions behind the scenes and is completely separate from any other Git version control system you might be using for the project as a whole.

However, this Git snapshot will recognize the .gitignore file defined in the project. This means if there are any artifacts (files, dependencies, etc.) larger than 50 MB stored directly in your project filesystem, make sure to add those files or folders to .gitignore so that they are not recorded as part of the snapshot. This ensures that the experiment or model environment is truly isolated and does not inherit dependencies that have been previously installed in the project workspace.

By default, each project is created with the following .gitignore file:

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

Augment this file to include any extra dependencies you have installed in your project workspace to ensure a truly isolated workspace for each model or experiment.

Multiple .gitignore files

A project can include multiple .gitignore files. However, there can only be one .git directory in the project, located at the project root, /home/cdsw/.git, otherwise Experiment and Model deployment fails.

If you create a blank project, and then want to clone a repo into it, clone a single project to the root of the workspace (/home/cdsw/.git) to ensure that Experiments and Models work.

Build Image

Once the code snapshot is available, Cloudera Data Science Workbench creates a new Docker image with a copy of the snapshot.

This new image is based off the project's designated default engine image (configured at Project Settings Engine). The image environment can be customized by using environmental variables and a build script that specifies which packages should be included in the new image.

Environmental Variables

Previously (CDSW 1.7.1 and lower), the environment variables set at the site admin level and project level did not automatically get pulled into the builds created for models and experiments. They needed to be explicitly coded into the cdsw-build.sh file. With CDSW 1.7.2 and higher, experiments and models will automatically inherit these admin and project-level environment variables.

Note that custom mounts or environment variables configured in cdsw.conf (such as NO_PROXY, HTTP(S)_PROXY, etc.) are still not passed to the container builds for experiments and models (even though they are applied to sessions, jobs, and deployed models/experiments).

For more information, see AWS Account Requirements.

Build Script - cdsw-build.sh

As part of the Docker build process, Cloudera Data Science Workbench runs a build script called cdsw-build.sh file. You can use this file to customize the image environment by specifying any dependencies to be installed for the code to run successfully. One advantage to this approach is that you now have the flexibility to use different tools and libraries in each consecutive training run. Just modify the build script as per your requirements each time you need to test a new library or even different versions of a library.

Important:

- The cdsw-build.sh script does not exist by default -- it has to be created by you within each project as needed.
- The name of the file is not customizable. It must be called cdsw-build.sh.

The following sections demonstrate how to specify dependencies in Python and R projects so that they are included in the build process for models and experiments.

Python 3

For Python, create a requirements.txt file in your project with a list of packages that must be installed. For example:

Figure 1: requirements.txt

```
beautifulsoup4==4.6.0
seaborn==0.7.1
```

Then, create a cdsw-build.sh file in your project and include the following command to install the dependencies listed in requirements.txt.

Figure 2: cdsw-build.sh

```
pip3 install -r requirements.txt
```

Now, when cdsw-build.sh is run as part of the build process, it will install the beautifulsoup4 and seaborn packages to the new image built for the experiment/model.

R

For R, create a script called install.R with the list of packages that must be installed. For example:

Figure 3: install.R

```
install.packages(repos="https://cloud.r-project.org", c("tidyr",
 "stringr"))
```

Then, create a cdsw-build.sh file in your project and include the following command to run inst all.R.

Figure 4: cdsw-build.sh

```
Rscript install.R
```

Now, when cdsw-build.sh is run as part of the build process, it will install the tidyr and stringr packages to the new image built for the experiment/model.

If you do not specify a build script, the build process will still run to completion, but the Docker image will not have any additional dependencies installed. At the end of the build process, the built image is then pushed to an internal Docker registry so that it can be made available to all the Cloudera Data Science Workbench hosts. This push is largely transparent to the end user.

Note: If you want to test your code in an interactive session before you run an experiment or deploy a model, run the cdsw-build.sh script directly in the workbench. This will allow you to test code in an engine environment that is similar to one that will eventually be built by the model/experiment build process.

Run Experiment / Deploy Model

Once the Docker image has been built and pushed to the internal registry, the experiment/model can now be executed within this isolated environment.

In case of experiments, you can track live progress as the experiment executes in the experiment's Session tab.

Unlike experiments, models do not display live execution progress in a console. Behind the scenes, Cloudera Data Science Workbench will move on to deploying the model in a serving environment based on the computing resources and replicas you requested. Once deployed you can go to the model's Monitoring page to view statistics on the number of requests served/dropped and stderr/stdout logs for the model replicas.

Creating and Deploying a Model (QuickStart)

Using Cloudera Data Science Workbench, you can create any function within a script and deploy it to a REST API. In a machine learning project, this will typically be a predict function that will accept an input and return a prediction based on the model's parameters.

About this task

For the purpose of this quick start demo we are going to create a very simple function that adds two numbers and deploy it as a model that returns the sum of the numbers. This function will accept two numbers in JSON format as input and return the sum.

Procedure

1. Create a new project. Note that models are always created within the context of a project.
2. Click Open Workbench and launch a new Python 3 session.
3. Create a new file within the project called add_numbers.py. This is the file where we define the function that will be called when the model is run. For example:

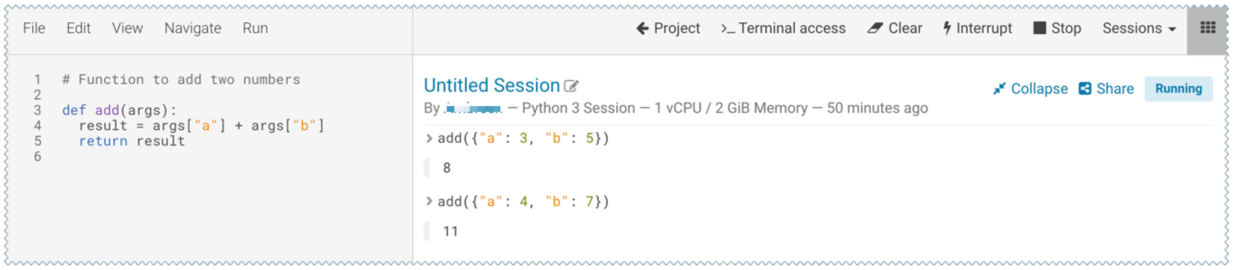
   Note: In practice, do not assume that users calling the model will provide input in the correct format or enter good values. Always perform input validation.

   add_numbers.py

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

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

   ```
   add({"a": 3, "b": 5})
   ```



```
# Function to add two numbers

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

By [redacted] — Python 3 Session — 1 vCPU / 2 GiB Memory — 50 minutes ago

```
> add({"a": 3, "b": 5})
8
> add({"a": 4, "b": 7})
11
```

5. Deploy the add function to a REST endpoint.

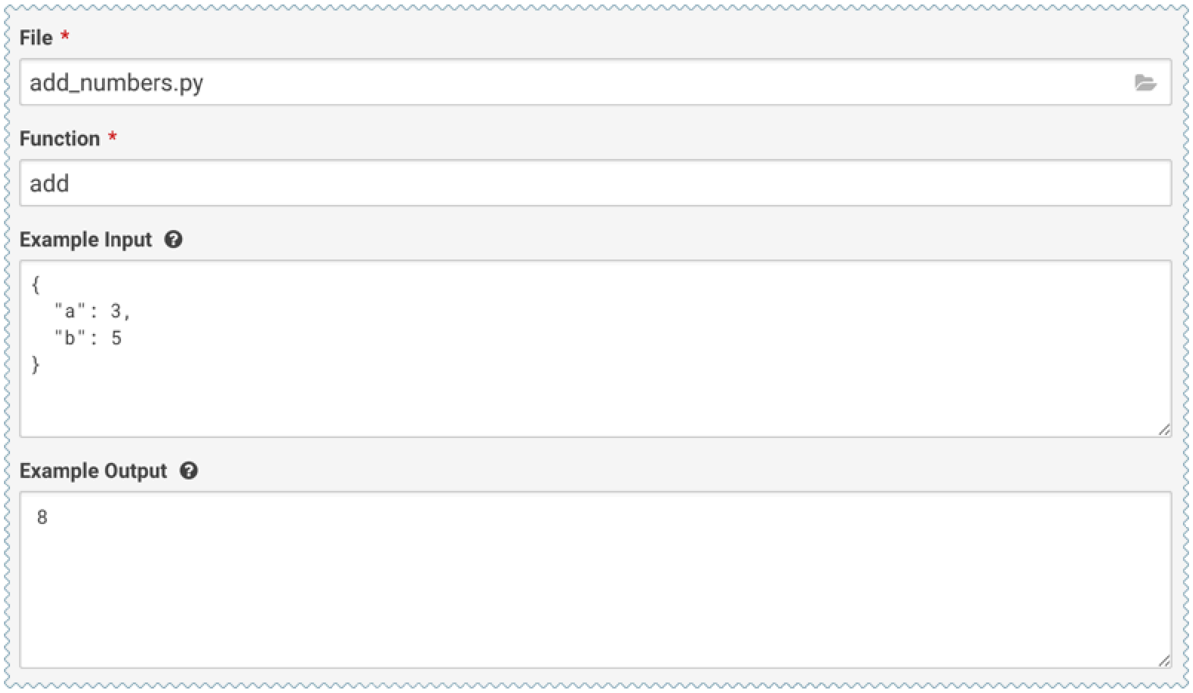
a) Go to the project Overview page.

b) Click Models New Model .

c) Give the model a Name and Description.

d) Enter details about the model that you want to build. In this case:

- File: add_numbers.py
- Function: add
- Example Input: {"a": 3, "b": 5}
- Example Output: 8



File *

add_numbers.py

Function *

add

Example Input

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

Example Output

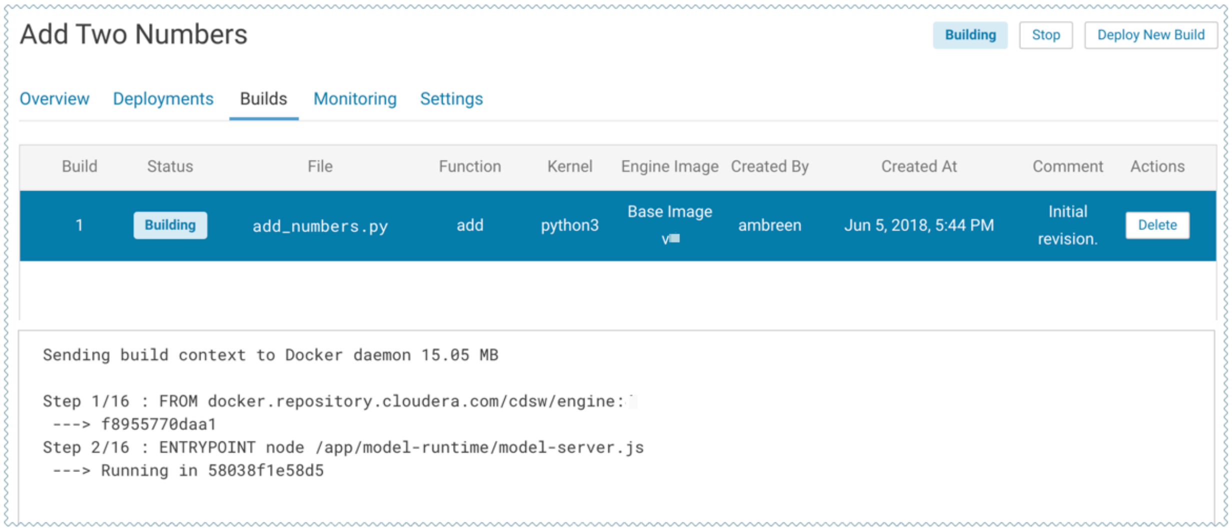
```
8
```

e) Select the resources needed to run this model, including any replicas for load balancing.

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

f) Click Deploy Model.

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



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

```
Sending build context to Docker daemon 15.05 MB

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

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

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

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

Calling a Model

This section lists some requirements for model requests and how to test a model using Cloudera Data Science Workbench.

(Requirement) JSON for Model Requests/Responses

Every model function in Cloudera Data Science Workbench takes a single argument in the form of a JSON-encoded object, and returns another JSON-encoded object as output. This format ensures compatibility with any application accessing the model using the API, and gives you the flexibility to define how JSON data types map to your model's datatypes.

Model Requests

When making calls to a model, keep in mind that JSON is not suitable for very large requests and has high overhead for binary objects such as images or video. Consider calling the model with a reference to the image or video such as a URL instead of the object itself. Requests to models should not be more than 5 MB in size. Performance may degrade and memory usage increase for larger requests.

Note: In Cloudera Data Science Workbench 1.4.0, model request sizes were limited to 100 KB. With version 1.4.2 (and higher), this limit has been increased to 5 MB. To take advantage of this higher threshold, you will need to upgrade to version 1.4.2 (or higher) and rebuild your existing models.

Ensure that the JSON request represents all objects in the request or response of a model call. For example, JSON does not natively support dates. In such cases consider passing dates as strings, for example in ISO-8601 format, instead.

For a simple example of how to pass JSON arguments to the model function and make calls to deployed model, see Creating and Deploying a Model (QuickStart)..

Model Responses

Models return responses in the form of a JSON-encoded object. Model response times depend on how long it takes the model function to perform the computation needed to return a prediction. Model replicas can only process one request at a time. Concurrent requests are queued until a replica is available to process them.

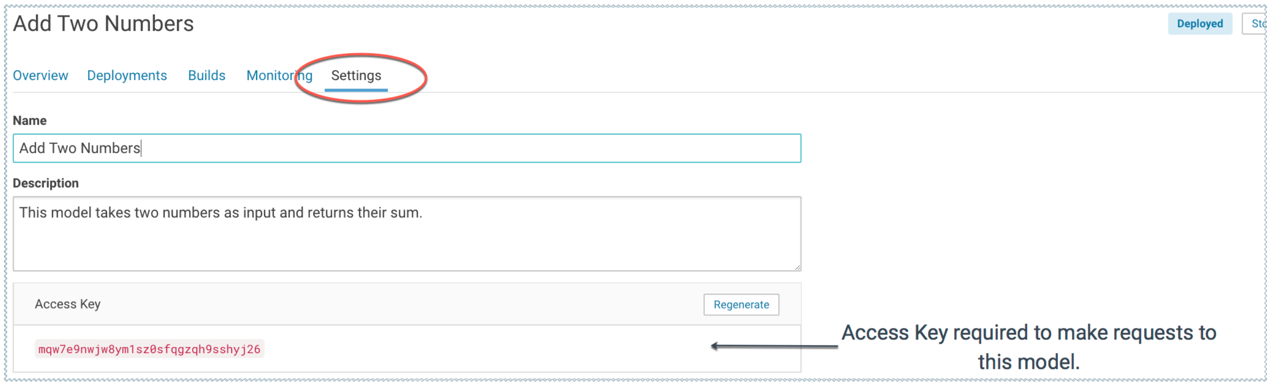
When Cloudera Data Science Workbench receives a call request for a model, it attempts to find a free replica that can answer the call. If the first arbitrarily selected replica is busy, Cloudera Data Science Workbench will keep trying to contact a free replica for 30 seconds. If no replica is available, Cloudera Data Science Workbench will return a mode l.busy error with HTTP status code 429 (Too Many Requests). If you see such errors, re-deploy the model build with a higher number of replicas.

(Requirement) Access Key

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

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

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



Name: Add Two Numbers

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

Access Key: mqw7e9nwjw8ym1sz0sfqgzqh9sshyj26

Important:

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

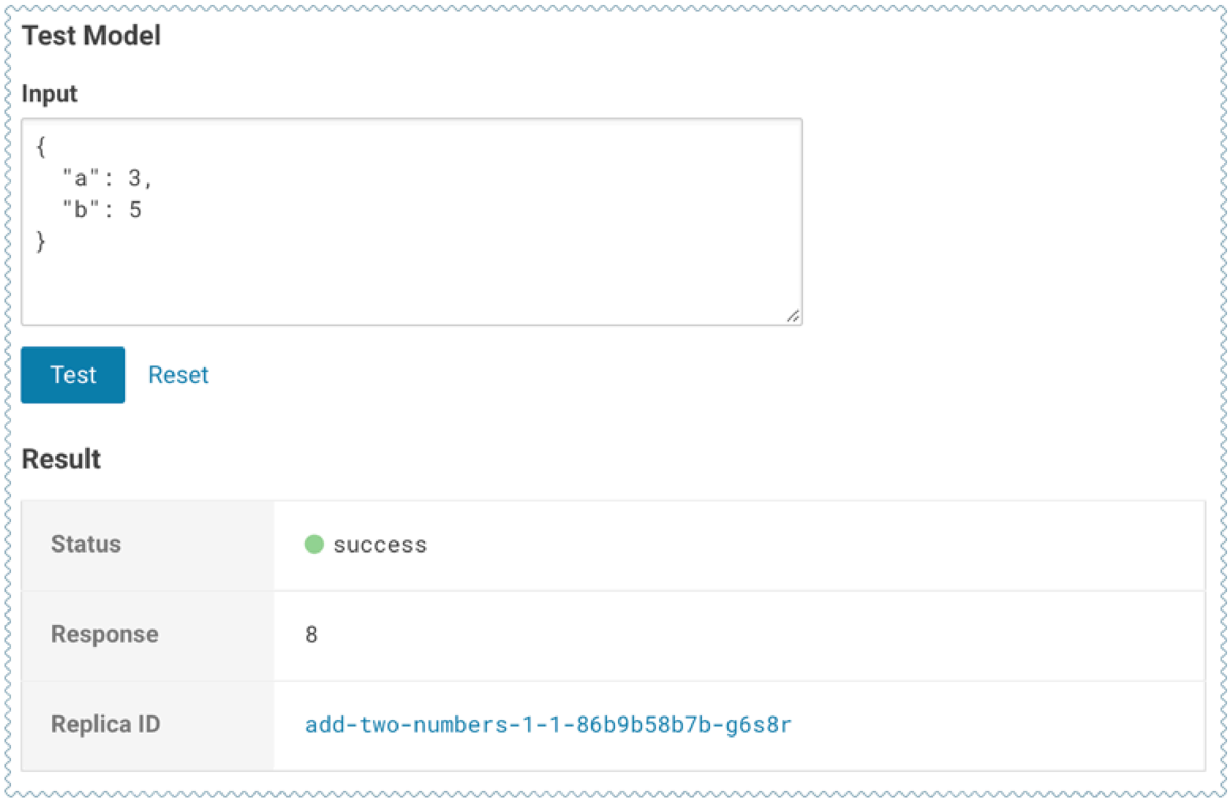
Alternatively, you can use this mechanism to revoke access to a model by regenerating the access key. Anyone with an older version of the key will not be able to make calls to the model.

Testing Calls to a Model

Cloudera Data Science Workbench provides two ways to test calls to a model:

- Test Model Widget

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



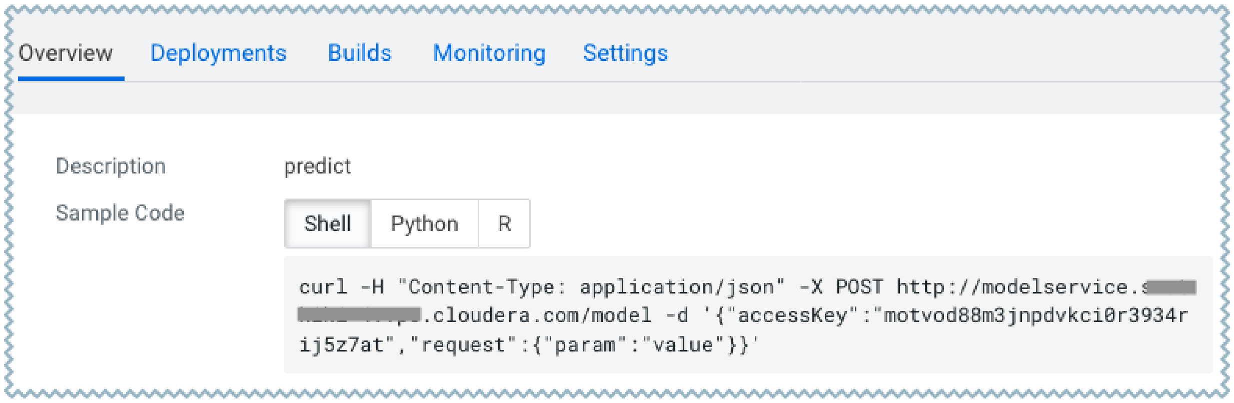
```
{
  "a": 3,
  "b": 5
}
```

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

- Sample Request Strings

  On the model Overview page, Cloudera Data Science Workbench also provides sample curl and POST request strings that you can use to test calls to the model. Copy/paste the curl request directly into a Terminal to test the call.

  Note that these sample requests already include the example input values you entered while building the model, and the access key required to query the model.



Description predict

Sample Code

```
curl -H "Content-Type: application/json" -X POST http://modelservice.[redacted].cloudera.com/model -d '{"accessKey":"motvod88m3jnpdvkci0r3934rij5z7at","request":{"param":"value"}}'
```

Model request timeout

You can set the model request timeout duration to a custom value. The default value is 30 seconds. The timeout can be changed if model requests might take more than 30 seconds. You need to have Admin access to Cloudera Manager to make this change.

To set the timeout value:

1. In Cloudera Manager, select Clusters, and then select the CDSW service.
2. In the Configuration tab, enter cdsw.properties in the Filters box.
3. In Application Advanced Configuration Snippet (Safety Valve) for cdsw.properties, enter the value in seconds (for example, 100):

```
MODEL_REQUEST_TIMEOUT_SECONDS=100
```

Updating Active Models

A model that is in the Deploying, Deployed, or Stopping stages.

You can make changes to a model even after it has been deployed and is actively serving requests. Depending on business factors and changing resource requirements, such changes will likely range from changes to the model code itself, to simply modifying the number of CPU/GPUs requested for the model. In addition, you can also stop and restart active models.

Depending on your requirement, you can perform one of the following actions:

Re-deploy an Existing Build

Re-deploying a model involves re-publishing a previously-deployed model in a new serving environment - this is, with an updated number of replicas or memory/CPU/GPU allocation.

About this task

For example, circumstances that require a re-deployment might include:

- An active model that previously requested a large number of CPUs/GPUs that are not being used efficiently.
- An active model that is dropping requests because it is falling short of replicas.
- An active model needs to be rolled back to one of its previous versions.

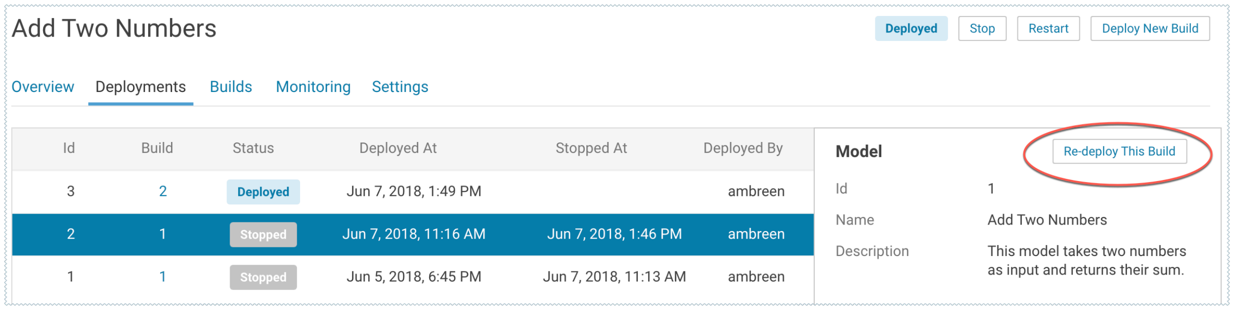
Warning: Currently, Cloudera Data Science Workbench only allows one active deployment per model. This means when you re-deploy a build, the current active deployment will go offline until the re-deployment process is complete and the new deployment is ready to receive requests. Prepare for model downtime accordingly.

To re-deploy an existing model:

Procedure

1. Go to the model Overview page.
2. Click Deployments.
3. Select the version you want to deploy and click Re-deploy this Build.

   Note: Select the version you want to deploy and click Re-deploy this Build.



Id	Build	Status	Deployed At	Stopped At	Deployed By
3	2	Deployed	Jun 7, 2018, 1:49 PM		ambreen
2	1	Stopped	Jun 7, 2018, 11:16 AM	Jun 7, 2018, 1:46 PM	ambreen
1	1	Stopped	Jun 5, 2018, 6:45 PM	Jun 7, 2018, 11:13 AM	ambreen

Id	1
Name	Add Two Numbers
Description	This model takes two numbers as input and returns their sum.

4. Modify the model serving environment as needed.

5. Click Deploy Model.

Deploy a New Build for a Model

Deploying a new build for a model involves both, re-building the Docker image for the model, and deploying this new build.

About this task

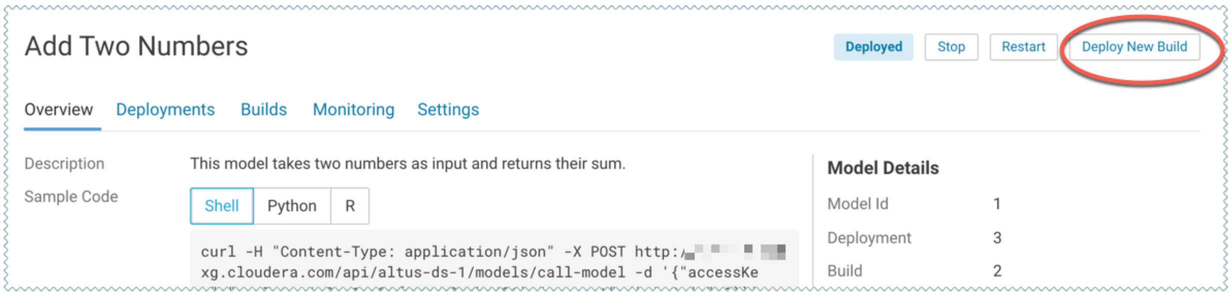
Note that this is not required if you only need to update the resources allocated to the model. As an example, changes that require a new build might include:

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

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

Procedure

1. Go to the model Overview page.
2. Click Deploy New Build.



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

Sample Code

```
curl -H "Content-Type: application/json" -X POST http:/[redacted]
xg.cloudera.com/api/altus-ds-1/models/call-model -d '{"accessKe
```

Model Details

Model Id	1
Deployment	3
Build	2

3. Complete the form and click Deploy Model.

Stop a Model

You can stop a model from the Overview page.

Procedure

1. To stop a model (all replicas), go to the model Overview page and click Stop.
2. Click OK to confirm.

Restart a Model

You can restart a model from the Overview page.

About this task

Restarting a model does not let you make any code changes to the model. It should primarily be used as a way to quickly re-initialize or re-connect to resources.

Procedure

1. To restart a model (all replicas), go to the model Overview page and click Restart.
2. Click OK to confirm.

Securing Models using Model API Key

You can prevent unauthorized access to your models by requiring the user to specify a Model API Key in the "Authorization" header of your model HTTP request. This topic covers how to create, test, and use a Model API Key in Cloudera Data Science Workbench.

The Model API key governs the authentication part of the process and the authorization is based on what privileges the users already have in terms of the project that they are a part of. For example, if a user/application has read-only access to a project, then the authorization is based on their current access-level to the project, which is "read-only". If the users have been authenticated to a project, then they can make a request to a model with the Model API Key. This is different to the previously described Access Key, which is only used to identify which model should serve a request.

Enabling Authentication

Restricting access using Model API Keys is an optional feature. By default, the Enable Authentication option is turned on. However, it is turned off by default for the existing models for backward compatibility. You can enable authentication for all your existing models.

Procedure

To enable authentication, go to Projects Models Settings and check the Enable Authentication option.

Note: It can take up to five minutes by the system to update.

Generating a Model API Key

If you have enabled authentication, then you need a Model API Key to call a model. If you are not a collaborator on a particular project, then you cannot access the models within that project using the Model API Key that you generate. You need to be added as a collaborator by the admin or the owner of the project to use the Model API Key to access a model.

About this task

There are two types of API keys used in Cloudera Data Science Workbench:

- API Key: This is used in the CDSW-specific internal APIs for CLI automation. This can't be deleted and neither does it expire. This API Key is not required when sending requests to a model.
- Model API Key: These are used to authenticate requests to a model. You can choose the expiration period and delete them when no longer needed.

You can generate more than one Model API Key to use with your model, depending on the number of clients that you are using to call the models.

Procedure

1. Sign in to the Cloudera Data Science Workbench.
2. Click Settings from the left navigation pane.
3. On the **User Settings** page, click the API Keys tab.

4. Select an expiry date for the Model API Key, and click Create API keys.

   An API key is generated along with a Key ID. If you do not specify an expiry date, then the generated key is active for one year from the current date.

   Note:

   - The Model API Key is private and ephemeral. Copy the key and the corresponding key ID on to a secure location for future use before refreshing or leaving the page. If you fail to store the key before refreshing the page, then you can generate another key.
   - You can delete the Model API Keys that have expired or are no longer in use. It can take up to five minutes for the system to take effect.
5. To test the Model API Key:
   a) Navigate to your project and click Models from the left navigation pane.
   b) On the **Overview** page, paste the Model API Key in the Model API Key field that you had generated in the previous step and click Test.

      The test results, along with the HTTP response code and the Replica ID are displayed in the Results table.

      If the test fails and you see the following message, then you must get added as a collaborator on the respective project by the admin or the creator of the project:

```
"User APikey not authorized to access model": "Check APIKEY permissions
or model authentication permissions"
```

Managing Model API Keys

The admin user can access the list of all the users who are accessing the workspace and can delete the Model API Keys for a user.

About this task

To manage users and their keys:

Procedure

1. Sign in to Cloudera Data Science Workbench as an admin user.
2. From the left navigation pane, click Admin.

   The **Site Administration** page is displayed.
3. On the **Site Administration** page, click on the Users tab.
4. To delete a Model API Key for a particular user:
   a) Select the user for which you want to delete the Model API Key.

      A page containing the user's information is displayed.
   b) To delete a key, click Delete under the Action column corresponding to the Key ID.
   c) Click Delete all keys to delete all the keys for that user.

      Note:

      It can take up to five minutes by the system to update.

      As a non-admin user, you can delete your own Model API Key by navigating to Settings User Settings API Keys .

Enabling Model Metrics

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

Procedure

1. Sign in to the Cloudera Manager web UI.
2. Go to Clusters CDSW service Configurations .
3. Select the Enable Model Metrics Support option and click Save Changes.

   Cloudera Manager detects stale configuration and prompts for a restart.
4. Select the Re-deploy client configuration option and restart the CDSW service.

   It can typically take 10-20 minutes for the CDSW service to restart.

Tracking Model Metrics

Tracking model metrics without deploying a model

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

Metrics tracked in this way are stored in a local, in-memory datastore instead of the metrics database, and are forgotten when the session exits. You can access these metrics in the same session using the regular metrics API in the cdsw.py file.

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

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

- use_model_metrics.py
- predict_with_metrics.py

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

```
@cdsw.model_metrics
def predict(args):
  # Track the input.
  cdsw.track_metric("input", args)

  # If this model involved features, ie transformations of the
  # raw input, they could be tracked as well.
  # cdsw.track_metric("feature_vars", {"a":1,"b":23})

  petal_length = float(args.get('petal_length'))
  result = model.predict([[petal_length]])
  # Track the output.
  cdsw.track_metric("predict_result", result[0][0])
  return result[0][0]
```

You can directly call this function in a workbench session, as shown in the following example:

```
predict(
{"petal_length": 3}
)
```

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

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

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

Tracking metrics for deployed models

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

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

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

Usage Guidelines

This section calls out some important guidelines you should keep in mind when you start deploying models with Cloudera Data Science Workbench.

Model Code

Models in Cloudera Data Science Workbench are designed to run any code that is wrapped into a function.

This means you can potentially deploy a model that returns the result of a SELECT * query on a very large table. However, Cloudera strongly recommends against using the models feature for such use cases.

As a best practice, your models should be returning simple JSON responses in near-real time speeds (within a fraction of a second). If you have a long-running operation that requires extensive computing and takes more than 15 seconds to complete, consider using batch jobs instead.

Model Artifacts

Once you start building larger models, make sure you are storing these model artifacts in HDFS, S3, or any other external storage. Do not use the project filesystem to store large output artifacts.

In general, any project files larger than 50 MB must be part of your project's .gitignore file so that they are not included in snapshots for future experiments/model builds. Note that in case your models require resources that are stored outside the model itself, it is up to you to ensure that these resources are available and immutable as model replicas may be restarted at any time.

Resouce Consumption and Scaling

Models should be treated as any other long-running applications that are continuously consuming memory and computing resources. If you are unsure about your resource requirements when you first deploy the model, start with a single replica, monitor its usage, and scale as needed.

If you notice that your models are getting stuck in various stages of the deployment process, check the monitoring page to make sure that the cluster has sufficient resources to complete the deployment operation.

Security Considerations

As stated previously, models do not impose any limitations on the code they can run. Additionally, models run with the permissions of the user that creates the model (same as sessions and jobs). Therefore, be conscious of potential data leaks especially when querying underlying data sets to serve predictions.

Cloudera Data Science Workbench models are not public by default. Each model has an access key associated with it. Only users/applications who have this key can make calls to the model. Be careful with who has permission to view this key.

Cloudera Data Science Workbench also prints stderr/stdout logs from models to an output pane in the UI. Make sure you are not writing any sensitive information to these logs.

Deployment Considerations

Cloudera Data Science Workbench does not currently support high availability for models. Additionally, there can only be one active deployment per model at any given time. This means you should plan for model downtime if you want to deploy a new build of the model or re-deploy with more/less replicas.

Keep in mind that models that have been developed and trained using Cloudera Data Science Workbench are essentially Python/R code that can easily be persisted and exported to external environments using popular serialization formats such as Pickle, PMML, ONNX, and so on.

Model Training and Deployment - Iris Dataset

This topic uses Cloudera Data Science Workbench's built-in Python template project to walk you through an end-to-end example where we use experiments to develop and train a model, and then deploy it using Cloudera Data Science Workbench.

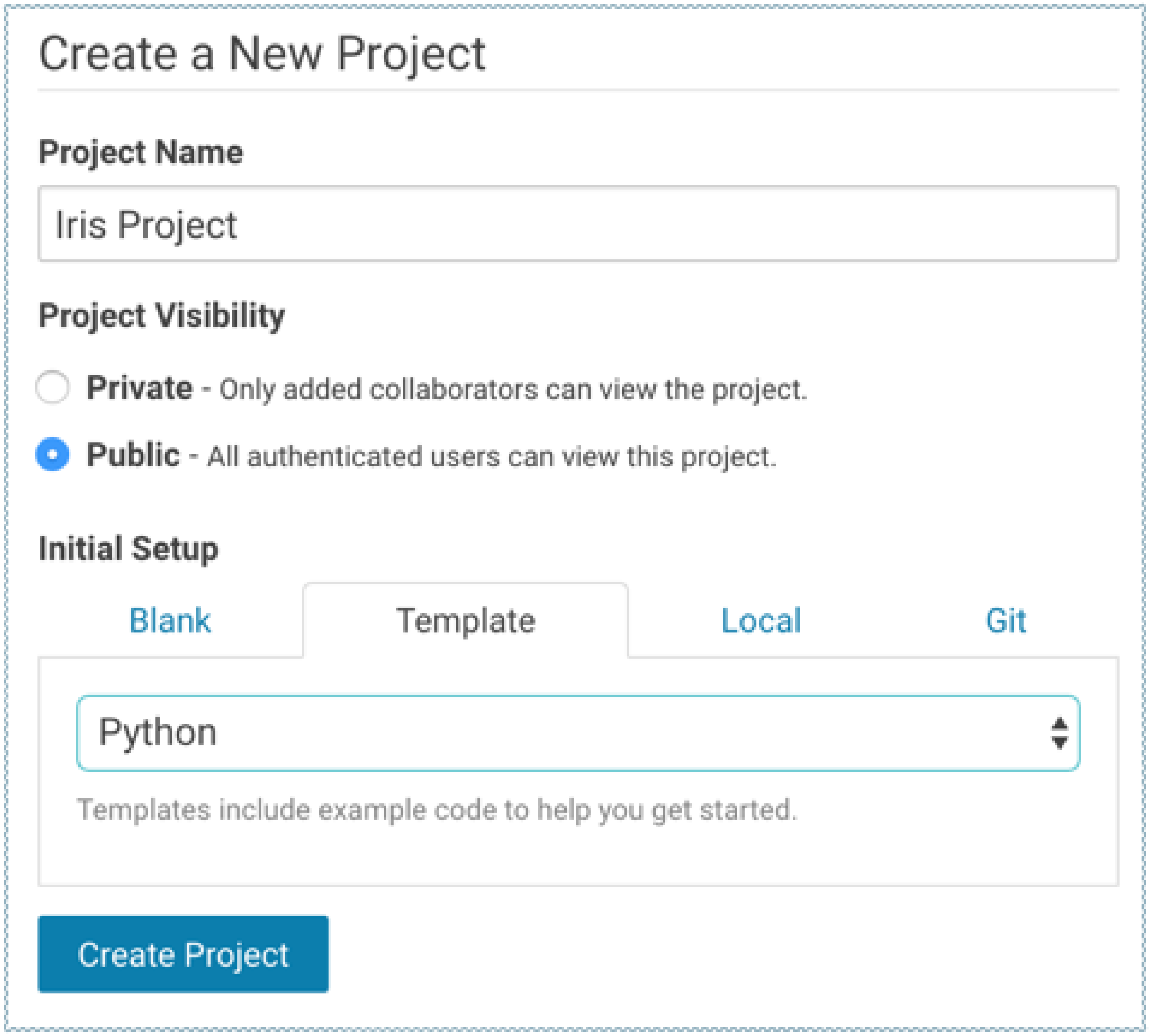
This example uses the canonical Iris dataset from Fisher and Anderson to build a model that predicts the width of a flower’s petal based on the petal's length.

Create a Project

The scripts for this example are available in the Python template project that ships with Cloudera Data Science Workbench.

Procedure

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



Create a New Project

Project Name

Iris Project

Project Visibility

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

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

Initial Setup

Python

Templates include example code to help you get started.

2. Once you've created the project, go to the project's Files page.

   The following files are used for the demo:

   - cdsw-build.sh - A custom build script used for models and experiments. Pip installs our dependencies, primarily the scikit-learn library.
   - fit.py - A model training example to be run as an experiment. Generates the model.pkl file that contains the fitted parameters of our model.
   - predict.py - A sample function to be deployed as a model. Uses model.pkl produced by fit.py to make predictions about petal width.

Train the Model

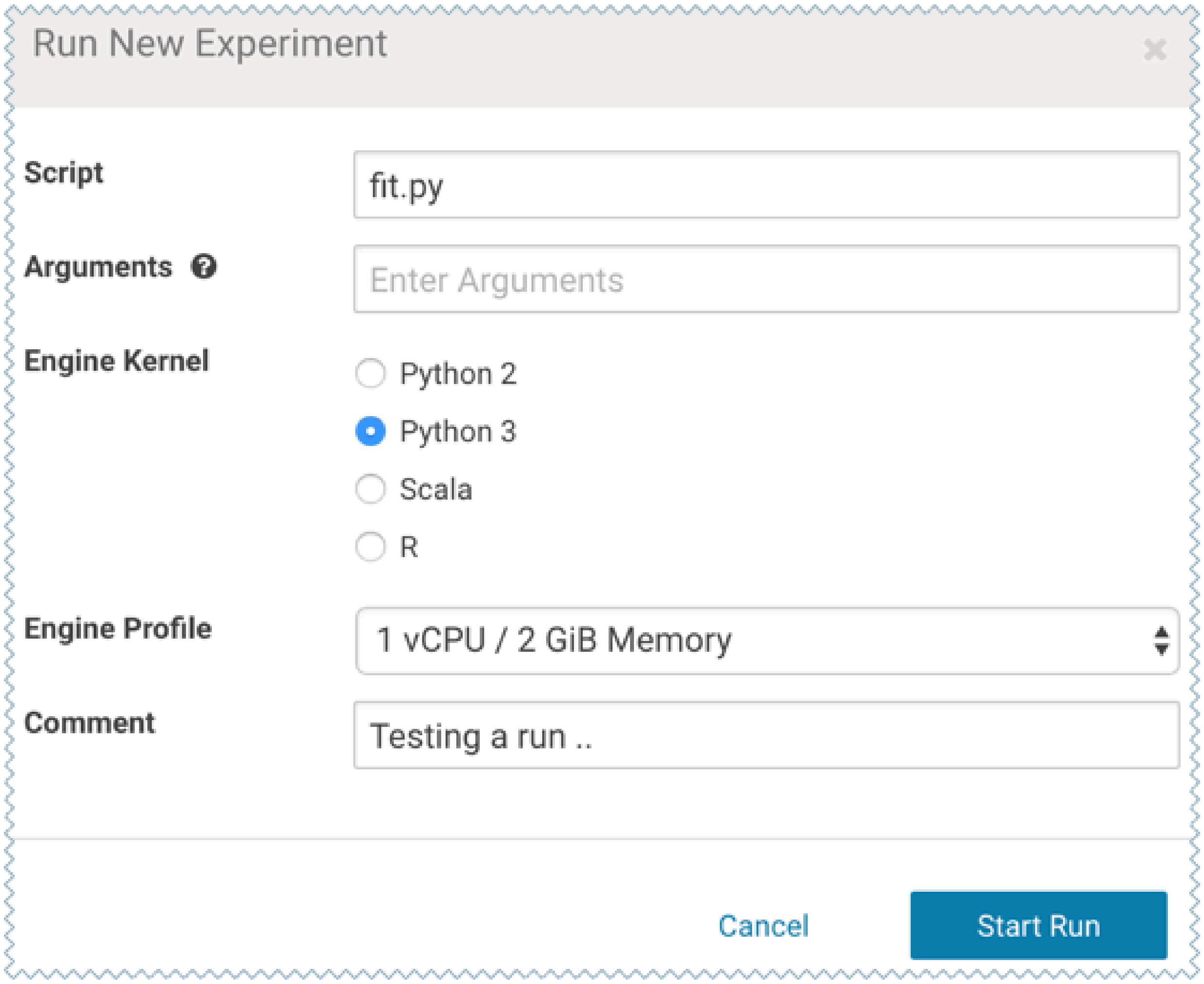
Run experiments using fit.py to develop a model. The fit.py script tracks metrics, mean squared error (MSE) and R2, to help compare the results of different experiments. It also writes the fitted model to a model.pkl file.

About this task

To run an experiment:

Procedure

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



Run New Experiment

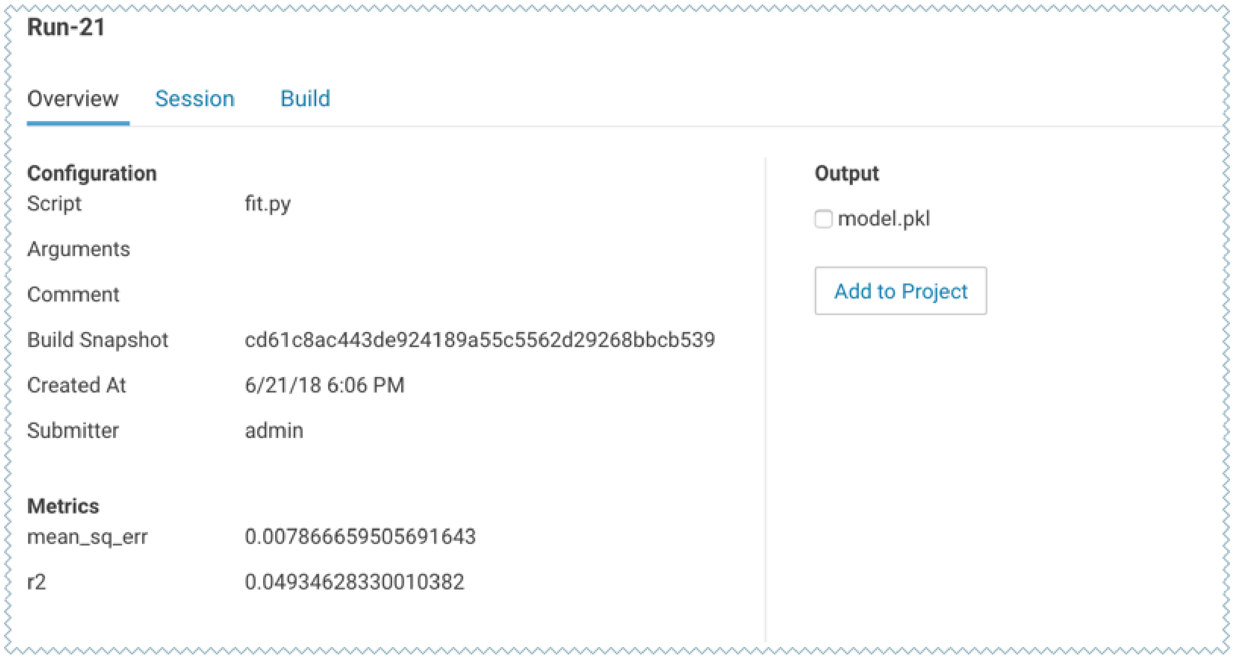
Script	fit.py
Arguments	Enter Arguments
Engine Kernel	Python 2 [✓ Python 3] Scala R
Engine Profile	1 vCPU / 2 GiB Memory
Comment	Testing a run ..

The new experiment should now show up on the Experiments table.

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

5. Once the experiment has completed successfully, go back to its Overview page.

   The tracked metrics show us that our test set had an MSE of ~0.0078 and an R2 of ~0.0493. For the purpose of this demo, let's consider this an accurate enough model to deploy and use for predictions.



Run-21

Configuration

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

Metrics

mean_sq_err	0.007866659505691643
r2	0.04934628330010382

6. Once you have finished training and comparing metrics from different experiments, go to the experiment that generated the best model. From the experiment's Overview page, select the model.pkl file and click Add to Project.

   This saves the model to the project filesystem, available on the project's Files page. We will now deploy this model as a REST API that can serve predictions.

Deploy the Model

To deploy the model we use the predict.py script from the Python template project.

Before you begin

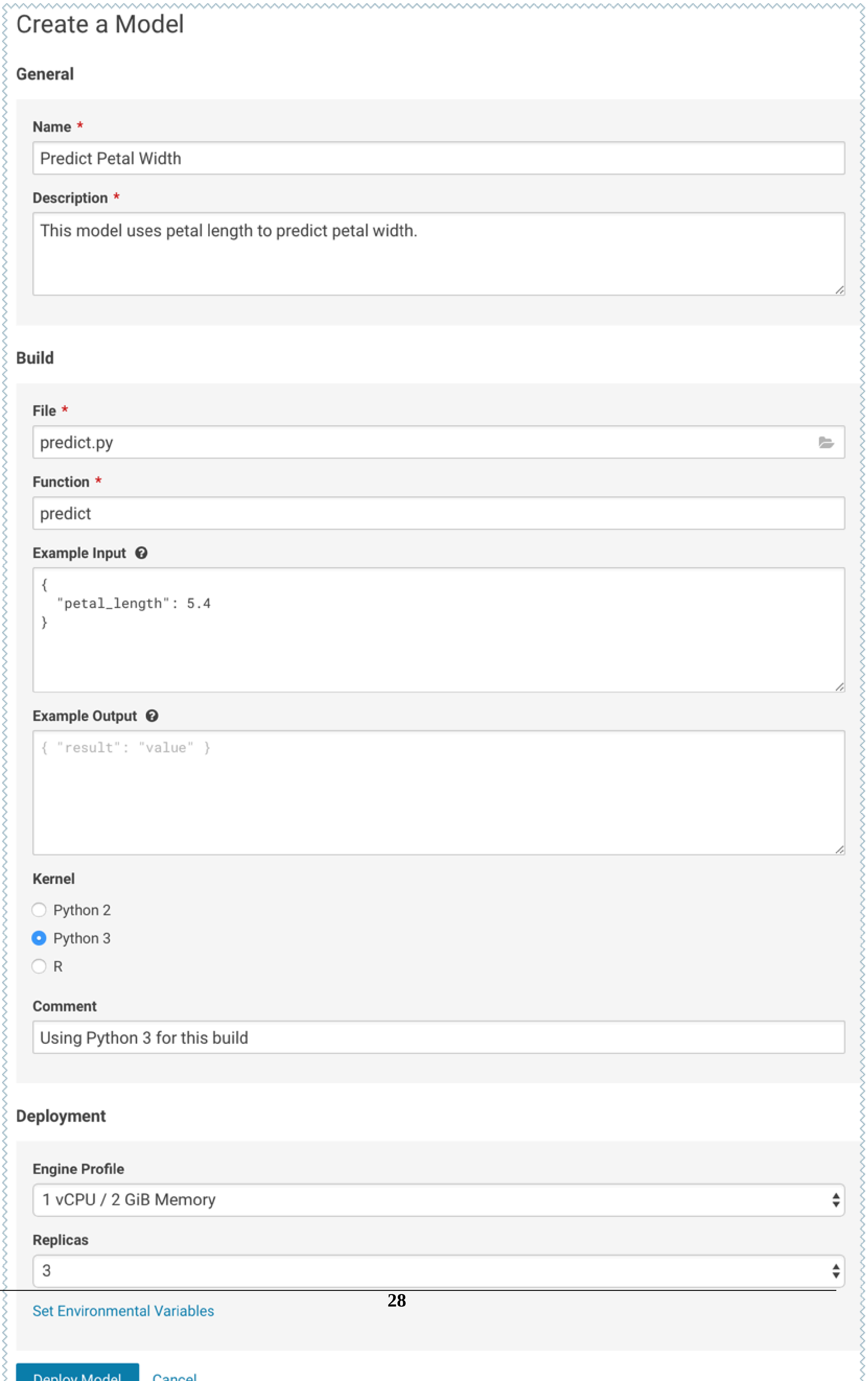
About this task

This script contains the predict function that accepts petal length as input and uses the model built in the previous step to predict petal width.

Procedure

1. Navigate to the Iris project's Overview Models page.

2. Click New Model and fill out the fields. Make sure you use the Python 3 kernel. For example:



Create a Model

General

Name *

Predict Petal Width

Description *

This model uses petal length to predict petal width.

Build

File *

predict.py

Function *

predict

Example Input

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

Example Output

```
{ "result": "value" }
```

Kernel

( ) Python 2
[✓ Python 3]
( ) R

Comment

Using Python 3 for this build

Deployment

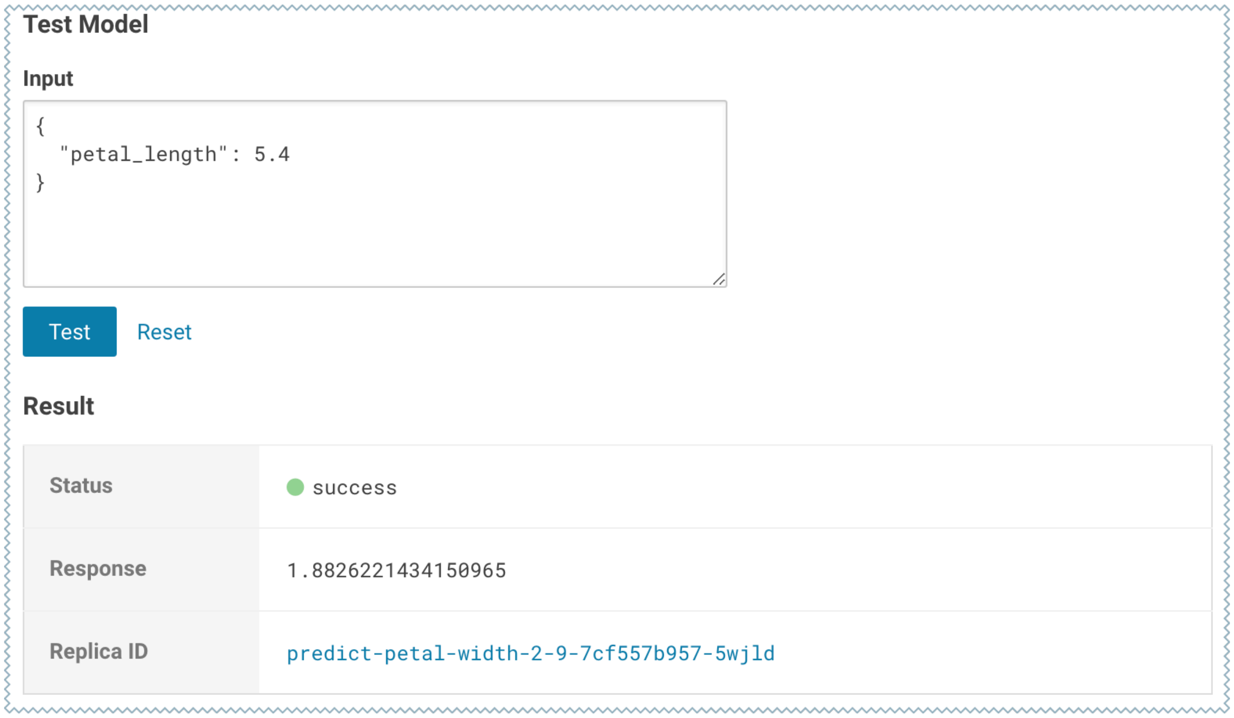
Engine Profile

1 vCPU / 2 GiB Memory

Replicas

3

3. Deploy the model.
4. Click on the model to go to its Overview page. As the model builds you can track progress on the Build page. Once deployed, you can see the replicas deployed on the Monitoring page.
5. To test the model, use the Test Model widget on the model's Overview page.



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

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

Model Monitoring and Administration

This topic describes how to monitor active models and some tasks related to general model administration.

Active Model - A model that is in the Deploying, Deployed, or Stopping stages.

Cloudera Data Science Workbench provides two ways to monitor active models.

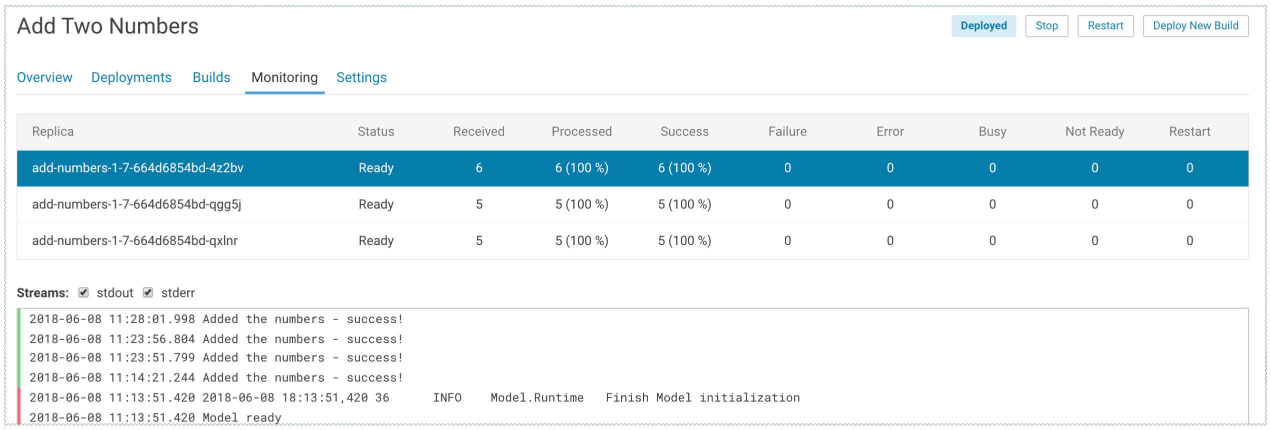
Monitoring Individual Models

When a model is deployed, Cloudera Data Science Workbench allows you to specify a number of replicas that will be deployed to serve requests.

Having more replicas means that your model should be able to serve more requests and is more resilient. However, we should not expect the distribution of requests to models to be completely evenly distributed across replicas. For instance, if you have one request that takes thirty seconds and two more immediate requests that take five seconds, we would expect the second replica to process more of the requests.

For each active model, you can monitor its replicas by going to the model's Monitoring page. On this page you can track the number of requests being served by each replica, success and failure rates, and their associated stderr and stdout logs. Depending on future resource requirements, you can increase or decrease the number of replicas by re-deploying the model.

The most recent logs are at the top of the pane (see image). stderr logs are displayed next to a red bar while stdout logs are by a green bar. Note that model logs and statistics are only preserved so long as the individual replica is active. When a replica restarts (for example, in case of bad input) the logs also start with a clean slate.



Replica	Status	Received	Processed	Success	Failure	Error	Busy	Not Ready	Restart
add-numbers-1-7-664d6854bd-4z2bv	Ready	6	6 (100 %)	6 (100 %)	0	0	0	0	0
add-numbers-1-7-664d6854bd-qgg5j	Ready	5	5 (100 %)	5 (100 %)	0	0	0	0	0
add-numbers-1-7-664d6854bd-qxlnr	Ready	5	5 (100 %)	5 (100 %)	0	0	0	0	0

Streams: ☑ stdout ☑ stderr

```
2018-06-08 11:28:01.998 Added the numbers - success!
2018-06-08 11:23:56.804 Added the numbers - success!
2018-06-08 11:23:51.799 Added the numbers - success!
2018-06-08 11:14:21.244 Added the numbers - success!
2018-06-08 11:13:51.420 2018-06-08 18:13:51,420 36      INFO    Model.Runtime   Finish Model initialization
2018-06-08 11:13:51.420 Model ready
```

Monitoring All Active Models

This section describes monitoring for all active models.

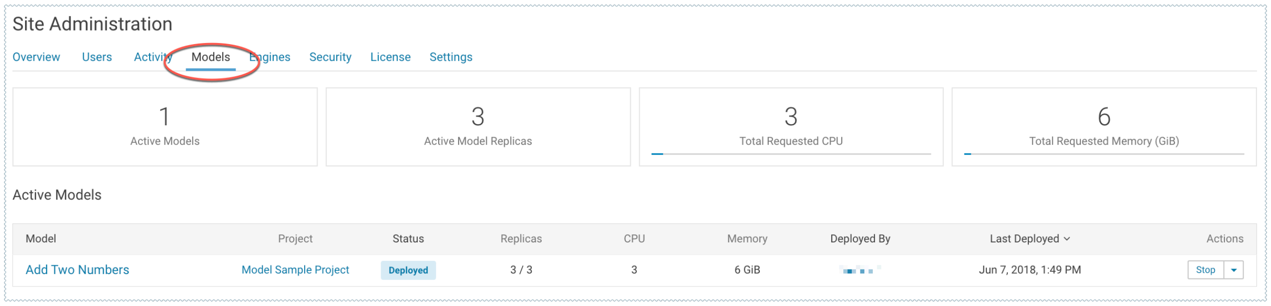
Before you begin

Required Role: Site Administrator

Procedure

To see a complete list of all the models that have been deployed on a deployment, and review resource usage across the deployment by models alone, go to Admin Models .

On this page, site administrators can also Stop/Restart/Rebuild any of the currently deployed models.



Model	Project	Status	Replicas	CPU	Memory	Deployed By	Last Deployed	Actions
Add Two Numbers	Model Sample Project	Deployed	3 / 3	3	6 GiB	[redacted]	Jun 7, 2018, 1:49 PM	Stop

Deleting a Model

Before you begin

- You must stop all active deployments before you delete a model. If not stopped, active models will continue serving requests and consuming resources even though they do not show up in Cloudera Data Science Workbench UI.
- Deleted models are not actually removed from disk. That is, this operation will not free up storage space.

About this task

Deleting a model removes all of the model's builds and its deployment history from Cloudera Data Science Workbench.

Procedure

To delete a model, go to the model Overview Settings and click Delete Model.

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

Disabling the Models Feature

Before you begin

Required Role: Site Administrator

About this task

Important: The feature flag mentioned here only hides the Models feature from the UI. It will not stop any active models that have already been deployed. Make sure you stop all active models from the Admin Models page before you disable the feature.

To disable this feature on your Cloudera Data Science Workbench deployment:

Procedure

1. Log in to Cloudera Data Science Workbench.
2. Click Admin Settings .
3. Under the Feature Flags section, disable the Enable users to create models. checkbox.

Debugging Issues with Models

This topic describes some common issues to watch out for during different stages of the model build and deployment process.

As a general rule, if your model spends too long in any of the afore-mentioned stages, check the resource consumption statistics for the cluster. When the cluster starts to run out of resources, often models will spend some time in a queue before they can be executed.

Resource consumption by active models on a deployment can be tracked by site administrators on the Admin Models page.

Building

Live progress for this stage can be tracked on the model's Build tab. It shows the details of the build process that creates a new Docker image for the model.

Potential issues:

- If you specified a custom build script (cdsw-build.sh), ensure that the commands inside the script complete successfully.
- If you are in an environment with restricted network connectivity, you might need to manually upload dependencies to your project and install them from local files.

Pushing

Once the model has been built, it is copied to an internal Docker registry to make it available to all the Cloudera Data Science Workbench hosts. Depending on network speeds, your model may spend some time in this stage.

Deploying

If you see issues occurring when Cloudera Data Science Workbench is attempting to start the model, use the following guidelines to begin troubleshooting.

- Make sure your model code works in a workbench session. To do this, launch a new session, run your model file, and then interactively call your target function with the input object. For a simple example, see the Model Quickstart.
- Ensure that you do not have any syntax errors. For python, make sure you have the kernel with the appropriate python version (Python 2 or Python 3) selected for the syntax you have used.

- Make sure that your cdsw-build.sh file provides a complete set of dependencies. Dependencies manually installed during a session on the workbench are not carried over to your model. This is to ensure a clean, isolated, build for each model.
- If your model accesses resources such as data on the CDH cluster or an external database make sure that those resources can accept the load your model may exert on them.

Deployed

Once a model is up and running, you can track some basic logs and statistics on the model's Monitoring page.

In case issues arise:

- Check that you are handling bad input from users. If your function throws an exception, Cloudera Data Science Workbench will restart your model to attempt to get back to a known good state. The user will see an unexpected model shutdown error.

  For most transient issues, model replicas will respond by restarting on their own before they actually crash. This auto-restart behavior should help keep the model online as you attempt to debug runtime issues.
- Make runtime troubleshooting easier by printing errors and output to stderr and stdout. You can catch these on each model's Monitoring tab. Be careful not to log sensitive data here.
- The Monitoring tab also displays the status of each replica and will show if the replica cannot be scheduled due to a lack of cluster resources. It will also display how many requests have been served/dropped by each replica.

  When Cloudera Data Science Workbench receives a call request for a model, it attempts to find a free replica that can answer the call. If the first arbitrarily selected replica is busy, Cloudera Data Science Workbench will keep trying to contact a free replica for 30 seconds. If no replica is available, Cloudera Data Science Workbench will return a model.busy error with HTTP status code 429 (Too Many Requests). If you see such errors, re-deploy the model build with a higher number of replicas.