H2O

Steam

Alpha Release

Table of Contents

Introduction

Steam is an "instant on" platform that streamlines the entire process of building and deploying applications. It is the industry's first data science hub that lets data scientists and developers collaboratively build, deploy, and refine predictive applications across large scale data sets. Data scientists can publish Python and R code as REST APIs and easily integrate with production applications.

Steam is available for download at h2o.ai/steam/.

User Management Overview

Before using Steam, it is important to understand User Management within your YARN environment. In Steam, User Management is supported in a PostgreSQL database. The User Management functions in Steam determine the level of access that users have for Steam features. The Steam database supports setup via CLI commands. Refer to the CLI Command Reference Appendix for a list of all available CLI commands.

For more information on Steam User Management, refer to the following sections.

- Terms
- Privileges/Access Control
- Authorization
- User Management Workflow

Terms

The following lists common terms used when describing Steam User Management.

- **Entities** represent *objects* in Steam. Examples of entities include Roles, Workgroups, Identities, Clusters, Projects, Models, and Services (engines).

- **Identities** represent *users* in Steam. Users sign in using an Identity, and then perform operations in Steam.

- **Permissions** determine what operations you can perform. Examples of permissions include *Manage Clusters*, *View Clusters*, *Manage Models*, *View Models*, and so on.

- **Privileges** determine the entities that you can perform operations on (i.e., data / access control).

Privileges/Access Control

Privileges are uniquely identified by the entity in question and the kind of privilege you have on the entity.

The following privileges are available on an entity:

- **Own** privileges allow you to share, view, edit, and delete entities.

- **Edit** privileges allow you to view and edit entities, but not share or delete them.

- **View** privileges allow you to view entities, but not share, edit, or delete them.

When you create an entity, you immediately *Own* it. You can then share this entity with others and award them either *Edit* or *View* privileges. Entities are allowed to have more than one owner, so you can also add additional owners to entities.

The following table lists the kind of privileges you need in order to perform specific operations on entities:

```
Entity              Own   Edit View
---------------------------------
Role
  Read              x     x     x
  Update            x     x
  Assign Permission x     x
  Delete            x
  Share             x

Workgroup
  Read              x     x     x
  Update            x     x
  Delete            x
  Share             x

Identity
  Read              x     x     x
  Assign Role       x     x
  Assign Workgroup  x     x
  Update            x     x
  Delete            x
  Share             x

Cluster
  Read              x     x     x
  Start/Stop        x

Project
  Read              x     x     x
  Assign Model      x     x
  Update            x     x
  Delete            x
  Share             x

Engine, Model
  Read              x     x     x
  Update            x     x
  Delete            x
  Share             x
```

Authorization

Permissions and privileges are set up using Roles and Workgroups, respectively.

- Identities cannot be linked directly to permissions. For that, you'll need Roles.

- Identities cannot be linked directly to privileges on entities. For that, you'll need Workgroups, i.e. when you share entities with others, you would be sharing those entities with workgroups, not individuals.

Roles

A **Role** is a named set of permissions. Roles allow you define a cohesive set of permissions into operational roles and then have multiple identities *play* those roles, regardless of access control. For example:

- a *Data Scientist* role can be composed of the permissions *View Clusters*, *Manage Models*, *View Models*.
- an *Operations* role can be composed of the permissions *View Models*, *View Services*, *Manage Services*,
- a *Manager* role can be composed of the permissions *Manage Roles*, *View Roles*, *Manage Workgroups*, *View Workgroups*

Workgroups

A **Workgroup** is a named set of identities. Workgroups allow you to form collections of identities for access control purposes. For example, a *Demand Forecasting* workgroup can be composed of all the users working on demand forecasting, regardless of their role. This workgroup can be then used to control access to all the clusters, projects, models and services that are used for demand forecasting.

User Management Workflow

The steps below provide a common workflow to follow when creating users. This workflow is followed in the example that follows.

1. Define roles based on operational needs.
2. Define workgroups based on data / access control needs.
3. Then add a new user:

   - Create the user's identity.
   - Associate the user with one or more roles.
   - Optionally, associate the user with one or more workgroups.

Next Steps

Now that you understand User Management, you can begin building and then setting up the H2O Scoring Service and Steam.

Installing, Starting, and H2O Using Steam on a YARN Cluster

This document is provided for external users and describes how to install, start, and use Steam on a YARN cluster. For instructions on using Steam on your local machine, refer to the Steam Standalone Installation document.

Requirements

- Web browser with an Internet connection
- Steam tar for OS X or Linux

  - available from h2o.ai/steam/
- JDK 1.7 or greater

- PostgreSQL 9.1 or greater

  - available from PostgreSQL.org
- H2O AutoML package (for example, **automl-hdp2.2.jar**)

Installation

Perform the following steps to install Steam.

Note: This installation should only be performed on a YARN edge node.

1. Go to h2o.ai/steam/ and download the Steam version for your platform (Linux or Mac). Be sure to accept the EULA.

2. Open a terminal window and SSH into your YARN edge node.

   ```
   ssh <user>@<yarn_edge_node>
   ```

   i. Untar the Steam package.
   Linux

   ```
   tar -xvf steamY-master-linux-amd64.tar.gz
   ```

OS X

```
tar -xvf steamY-master-darwin-amd64.tar.gz
```

Start the PostgreSQL Database

The command starts PostgreSQL. This should be started from the folder where PostgreSQL was installed.

```
postgres -D /usr/local/var/postgres
```

Create a User

This step create a new user for the Steam database and then create the database. The commands below only need to be performed once. The example below creates a steam **superuser** with a password `superuser` before creating the Steam database. Be sure to provide a secure password, and be sure to remember the password that you enter. This will be required each time you log in to Steam.

```
createuser -P steam
Enter password for new role: superuser
Enter it again: superuser
# Change directories to the Steam /var/master/scripts folder.
cd steam-master-darwin-amd64/var/master/scripts
./create-database.sh
```

Starting Steam, the Steam Scoring Service, and the Steam CLI

Perform the following steps to start Steam and the Steam Scoring Service. Note that two terminal windows will remain open: one for the Jetty server and one for Steam.

1. Change directories to your Steam directory, then set up the Jetty server using one of the following methods:

   Linux

```
cd steam-master-linux-amd64
java -jar var/master/assets/jetty-runner.jar var/master/assets/ROOT.war
```

OS X

```
cd steam-master-darwin-amd64
java -jar var/master/assets/jetty-runner.jar var/master/assets/ROOT.war
```

Note: The Jetty server defaults to port 8080. You can optionally provide a `--port` value for jetty-runner.jar.

2. Open another terminal window and ssh to the machine running YARN.

```
ssh <user>@<yarn_edge_node>
```

  i. Change directories to the Steam directory (either **steam-master-linux-amd64** or **steam-master-darwin-amd64**), then start the Steam master node. For example, the following commands will start Steam on 192.168.2.182. Note that the port value must match the port running the Jetty server, which defaults to 8080.

```
./steam serve master --superusern-name=superuser --superuser-password=superuse
r --compilation-service-address="192.168.2.182:8080"
```

Note: You can view all available options for starting Steam using `./steam help serve master`

You will see a message similar to the following when Steam starts successfully.

```
2016/04/28 13:34:56 steam v build 2016-04-28T20:15:00+0000
2016/04/28 13:34:56 Working directory: /home/seb/steam--linux-amd64/var/master
2016/04/28 13:34:56 WWW root: /home/seb/steam--linux-amd64/var/master/www
2016/04/28 13:34:57 Priming datastore for first time use...
2016/04/28 13:34:57 Datastore location: /home/seb/steam--linux-amd64/var/maste
r/db/steam.db
2016/04/28 13:34:57 Web server listening at 192.16.2.182:9000
2016/04/28 13:34:57 Point your web browser to http://192.16.2.182:9000/
```

  i. Open another terminal window. From within the Steam folder, log in to the machine running Steam. Use the password that you provided when you created superuser.

```
./steam login 192.168.2.182:8080 --username=superuser --password=superuser
```

   i. Run the following to verify that the CLI is working correctly.

```
./steam help
```

At this point, you can begin using Steam or continue adding more roles, workgroups, and users using the CLI. (Refer to Adding Roles, Workgroups, and Users.

Using Steam

In a Web browser, navigate to the Steam Web server (for example, http://192.16.2.182:9000).

Adding an Engine

An empty Steam UI will display. Before performing any tasks, you must first add an Asset. In this case, the **automl-hdp2.2.jar** file (or similar) provides the engine necessary for Steam to run AutoML jobs.

1. Click the **Assets** icon (![ASSETS]) on the left navigation panel, then select **Add Engine**.
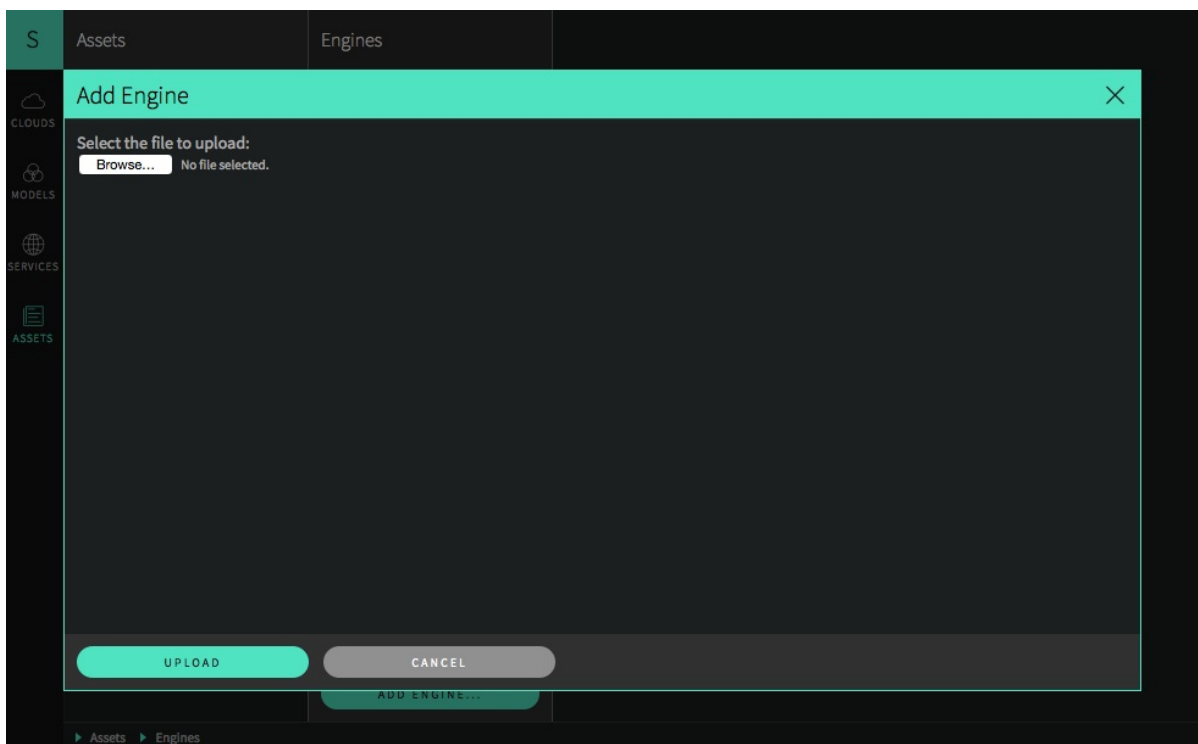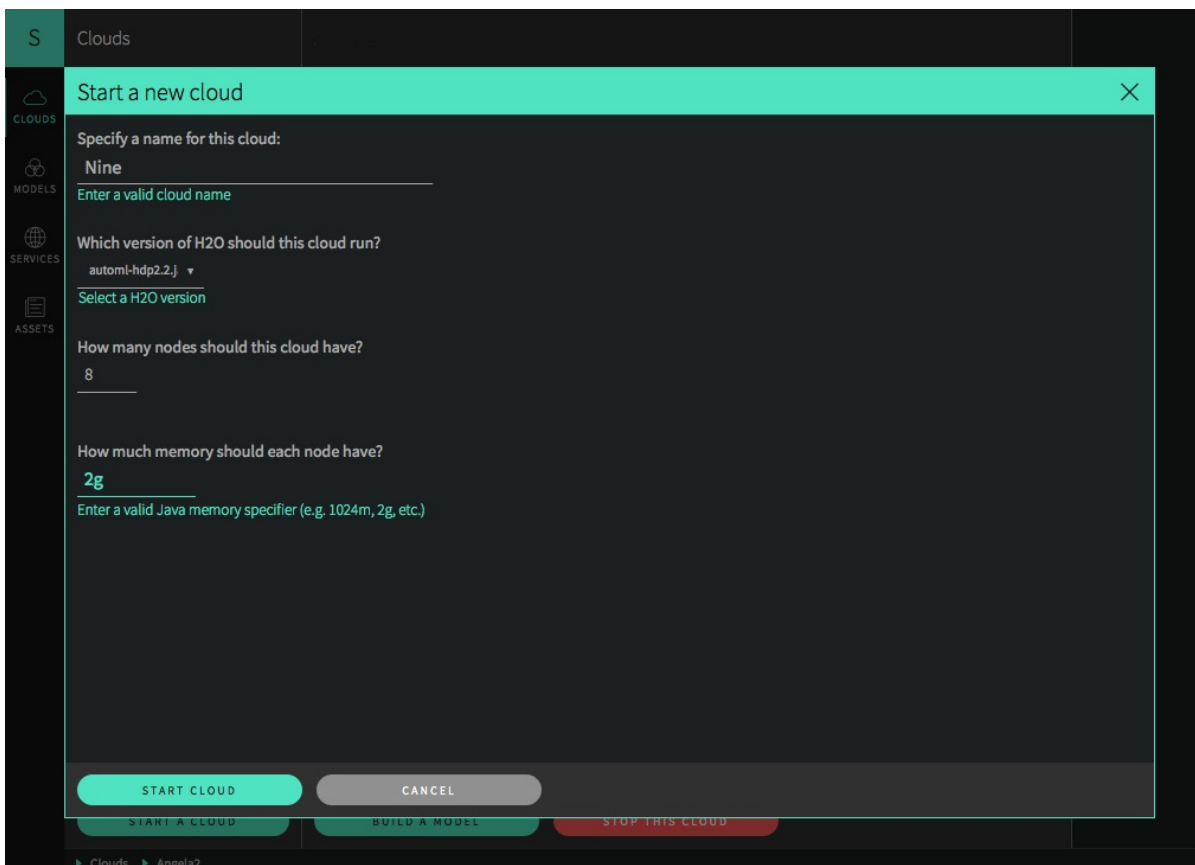
2. Browse to the **automl-hdp2.2.jar** file on your local machine, then click **Upload**.

Starting a Cluster

Cluster can be configured after the engine asset was successfully added.

1. Click the **Clusters** icon (CLOUDS) on the left navigation panel, then select **Start a Cluster**.

2. Enter/specify the following information to set up your cloud:

   a. A name for the cluster.

   b. The version of H2O that will run on the cloud.

   c. The number of nodes on the cloud.

   d. The amount of memory available on each node. Be sure to include the unit ("m" or "g").

3. Click **Start Cluster** when you are finished.

The Cluster Details page opens upon successful completion. This page shows the cluster configuration information and includes a link to the H2O Flow URL. From this page, you can begin building your model.

Note: You can view a stream of the cluster creation log in the terminal window that is running Steam. In the UI, Steam will respond with an error if the cloud configuration is incorrect (for example, if you specify more nodes than available on the cluster).

Adding a Model

Models can be created using Flow, R, or Python when H2O is initialized using this cluster. Once created, the model will automatically be visible in Steam.

Viewing Models

Click the **Models** icon ( ) on the left navigation panel to view models that were successfully created.

Deploying Models

After a model is built, the next step is to deploy the model in order to make/view predictions.

1. Click the **Models** icon () on the left navigation panel.

2. Select the model that you want to use, then click the **Deploy this Model** button on the bottom of the page.

3. Specify the port to use for the scoring service.

   Note: Steam will return an error if you specify a port that is already being used.

4. Click **Deploy** when you are done.

Making Predictions

Successfully deployed models can be viewed on the **Services** page. On this page, click the Endpoint link to launch the H2O Prediction Services UI. From this page, you can make predictions using one of the following methods:

- Specify input values based on column data from the original dataset

  OR

- Enter a query string using the format `field1=value1&field2=value2` (for example, `sepal_width=3&petal_len=5`)

Use the **Clear** button to clear all entries and begin a new prediction.

You can view additional statistics about the scoring service by clicking the **More Stats** button.

Refer to the H2O Scoring Service Builder document for more information.

Using Steam with H2O Flow

As with other H2O products, Flow can be used alongside Steam when performing machine learning tasks.

On the Cloud Details page, click the **Address** link to open H2O Flow in a new tab.
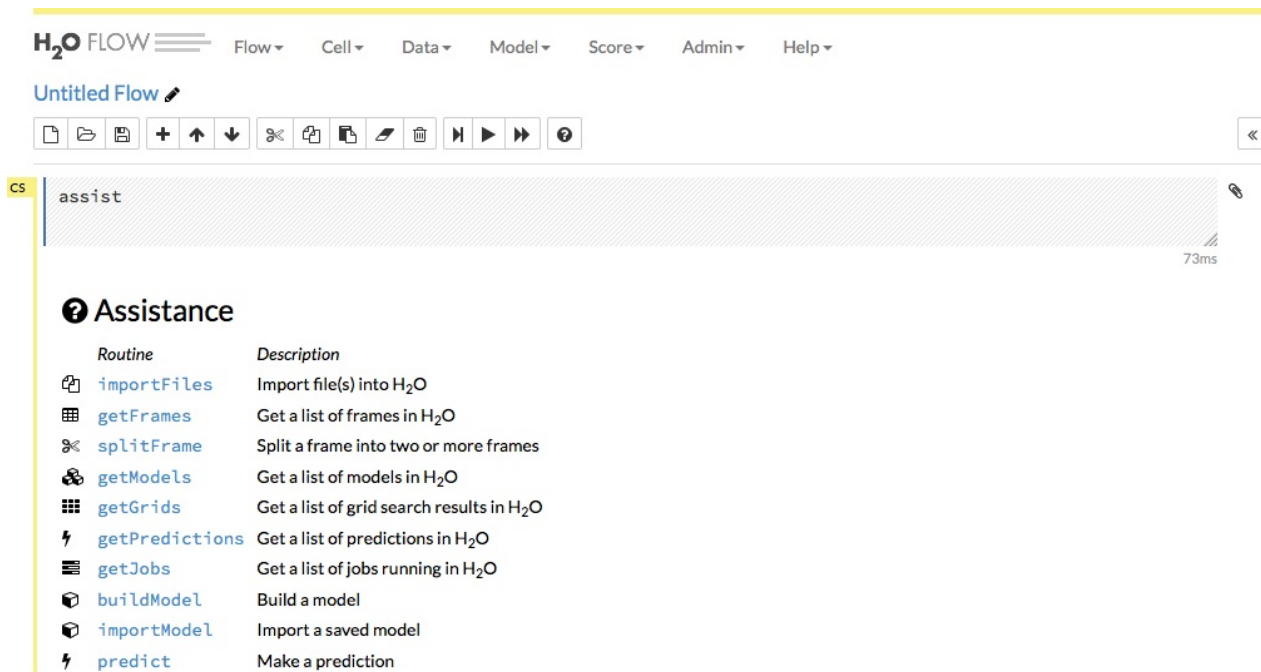
Note: Refer to the H2O Flow documentation for information on how to use Flow.

Stopping Steam

When you are finished, use the following process to safely shut down Steam:

1. On the Services page, stop all running services.

2. Stop all running clouds.

Steam Standalone Installation

This document describes how to use Steam without the need for a local running instance of YARN. These instructions will walk through the following procedures:

- Installing and starting Steam, the Compilation Service, and H2O
- Adding Roles, Workgroups, and Users to the database
- Building a simple model in Python (Optional for users who don't have an existing demo.)
- Deploying the model using Steam

During this demo, three terminal windows will remain open for the Steam, Scoring, and H2O services. A fourth terminal window will be used to run H2O commands in the Python or R example.

Finally, these steps were created using H2O version 3.8.2.8, and that version resides in a Downloads folder. Wherever used, this version number and path should be adjusted to match your version and path.

Requirements

- Web browser with an Internet connection
- JDK 1.7 or greater
- PostgreSQL 9.1 or greater

    - available from PostgreSQL.org
- Steam tar for Linux or OS X

    - available from h2o.ai/steam/
- H2O jar file

    - available from the H2O Download page

Optional

The following are required if you use a Python or R demo.

Python

- A dataset that will be used to generate a model. This demo uses the well-known iris.csv dataset with headers (available online), and the dataset is saved onto the desktop.
- Python 2.7

R

- A dataset that will be used to generate a model.
- Comprehensive R Archive Network (R). Available from https://cran.r-project.org/mirrors.html.

Starting Steam

This section describes how to set up and start Steam and start the Steam CLI for user management. Five terminal windows will be open the first time you run this setup; four terminal windows will be open for subsequent logins.

1. Go to h2o.ai/steam/ and download the Steam version for your platform (Linux or Mac). Be sure to accept the EULA.

2. Open a terminal window and untar the steamY binary. Note that the command below untars the OS X binary. Replace `darwin` with `linux` in the steps that follow to build on Linux.

   ```
   user$ tar xvf steamY-master-darwin-amd64.tar.gz
   ```

3. Open a second terminal window and start PostgreSQL. This should be started from the folder where PostgreSQL was installed.

   ```
   postgres -D /usr/local/var/postgres
   ```

   i. Open a third terminal window to create a new user for the Steam database and then create the database. The commands below only need to be performed once. The example below creates a steam **superuser** with a password `superuser` before creating the Steam database. Be sure to provide a secure password, and be sure to remember the password that you enter. This will be required each time you log in to Steam.

```
createuser -P steam
Enter password for new role: superuser
Enter it again: superuser
# Change directories to the Steam /var/master/scripts folder.
cd steam-master-darwin-amd64/var/master/scripts
./create-database.sh
```

i. Change directories to your Steam directory, and start the Jetty server.

```
user$ cd steam-master-darwin-amd64
user$ java -jar var/master/assets/jetty-runner.jar var/master/assets/ROOT.war
```

> *Note*: The Jetty server defaults to port 8080. You can optionally provide a `--port` value for **jetty-runner.jar**.

4. Open a fourth terminal window. From within the **steam-maseter-darwin-amd64** folder, start the Steam compilation and scoring service using the password that you provided in Step 2. This starts Steam on localhost:9000.

```
./steam serve master --superuser-name=superuser --superuser-password=superuser
```

> *Note*: This starts the Steam web service on `localhost:9000`, the compilation service on `localhost:8080` (same as the Jetty server), and the scoring service on `localhost`. You can change these using `--compilation-service-address=<ip_address:port>` and `--scoring-service-address=<ip_address>`. Use `./steam help serve master` or `./steam serve master -h` to view additional options.

5. Open a fifth terminal window. From within the Steam folder, log in to the machine running Steam (localhost:9000). Use the password that you provided in Step 2.

```
./steam login localhost:9000 --username=superuser --password=superuser
```

i. Run the following to verify that the CLI is working correctly.

```
./steam help
```

At this point, you can open a browser and navigate to localhost:9000. Note that you may be prompted to once more provide the login credentials supplied in Step 7.

The next section describes how to add additional users to the Steam database.

Adding Roles, Workgroups, and Users

The following example creates sample roles, workgroups, and users using the CLI. Refer to the CLI Command Reference Appendix for information about all of the commands available in the CLI. These commands are run from the terminal window used to log in to Steam (Step 7 above).

```
# Create engineer role and link that role to permissions
./steam create role engineer --desc="a default engineer role"
./steam link role engineer ViewModel ViewProject ViewWorkgroup

# Create data scientist role and link that role to permissions
./steam create role datascience --desc="a default data scientist role"
./steam link role datascience ManageProject ManageModel ViewCluster

# Create preparation and production workgroups
./steam create workgroup preparation --desc="data prep group"
./steam create workgroup production --desc="production group"

# Create two users - Bob and Jim
./steam create identity bob bobSpassword
./steam create identity jim j1mSpassword

# Link Bob to engineer role; link Jim to datascience role
./steam link identity bob role engineer
./steam link identity jim role datascience

# Link Bob to preparation workgroup; link Jim to production workgroup
./steam link identity bob workgroup preparation
./steam link identity jim workgroup production
```

Using H2O with Steam

Now that Steam is running and users are set up, this section describes how to use H2O with Steam.

1. Open another terminal window. Navigate to the folder with your H2O jar file and start H2O. This will create a one-node cluster on your local machine on port 54321.

   ```
   user$ cd ~/Downloads/h2o-3.8.2.8
   user$ java -jar h2o.jar
   ```

   i. Point your browser to the Steam URL, for example, http://localhost:9000/.

2. In the left pane, select the **Clusters** tab (selected by default), then click the **Connect To Cluster** button to setup Steam with H2O. Specify the IP address and port of the cluster currently running H2O (for example, localhost:54321), then click **Register Cluster**.
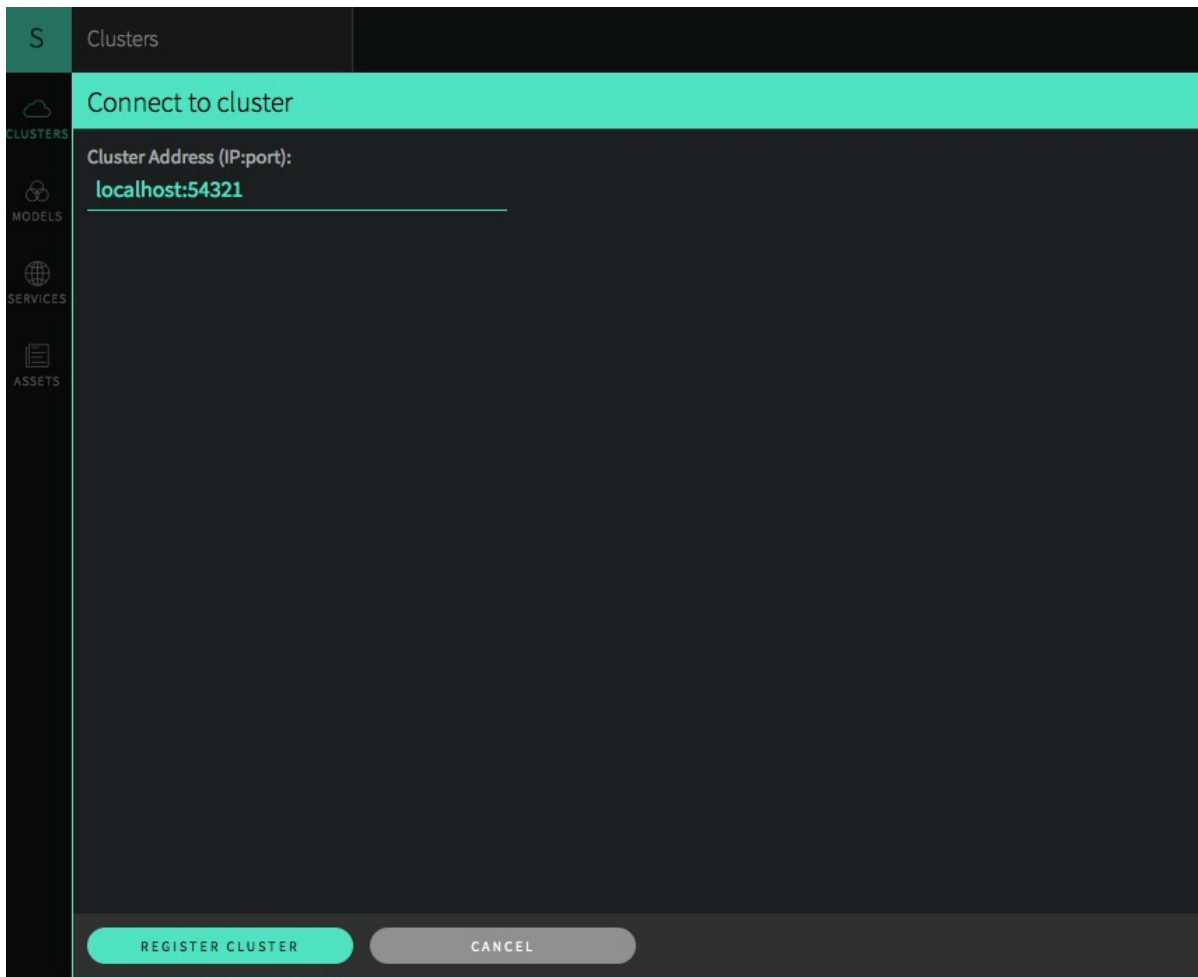
You are now ready to build a model on this cluster in Python.

> **Note**: After you connect to a cluster, click on **Cluster Details**, select your cluster, then click the **Address** link on that page to launch H2O Flow.

Building a Model in Python (Optional)

> **Notes**: This section can be skipped if you already have demo steps that you use in R, Python, or Flow. If you use another demo, be sure that you initialize H2O on your local cluster so that the data will be available in Steam.
>
> Additional demos for Python are available here.
>
> Demos for R are available here.
>
> A demo of Flow can be viewed here.

The steps below show how to build model using the Iris dataset and the GBM algorithm. The steps will be run using H2O in Python. Once created, the model can be deployed in Steam.

1. Open a terminal window. Change directories to the H2O folder, and start Python. Import the modules that will be used for this demo.

```
$ cd ~/Downloads/h2o-3.8.2.8
$ python
>>> import h2o
>>> from h2o.estimators.gbm import H2OGradientBoostingEstimator
```

   i. Initialize H2O using localhost and port 54321. (Note that if started Steam on a different machine, then replace `localhost` with the IP address of that machine.)

```
>>> h2o.init(ip="localhost", port=54321)
----------------------------    ------------------------------------
H2O cluster uptime:             2 minutes 37 seconds 168 milliseconds
H2O cluster version:            3.8.2.8
H2O cluster name:               user
H2O cluster total nodes:        1
H2O cluster total free memory:  3.35 GB
H2O cluster total cores:        8
H2O cluster allowed cores:      8
H2O cluster healthy:            True
H2O Connection ip:              127.0.0.1
H2O Connection port:            54321
H2O Connection proxy:
Python Version:                 2.7.9
----------------------------    ------------------------------------
```

   i. Upload the Iris dataset. Note that in this example, Python is running from the Downloads folder, and the Iris dataset is on the Desktop:

```
>>> df=h2o.upload_file("../../Desktop/iris.csv")
```

   i. Specify the configuration options to use when building a GBM model.

```
>>> gbm_regressor = H2OGradientBoostingEstimator(distribution="gaussian", ntre
es=10, max_depth=3, min_rows=2, learn_rate="0.2")
```

   i. Train the model using the Iris dataset (`df` object) and the GBM configuration options.

```
>>> gbm_regressor.train(x=range(1, df.ncol), y=0, training_frame=df)
```
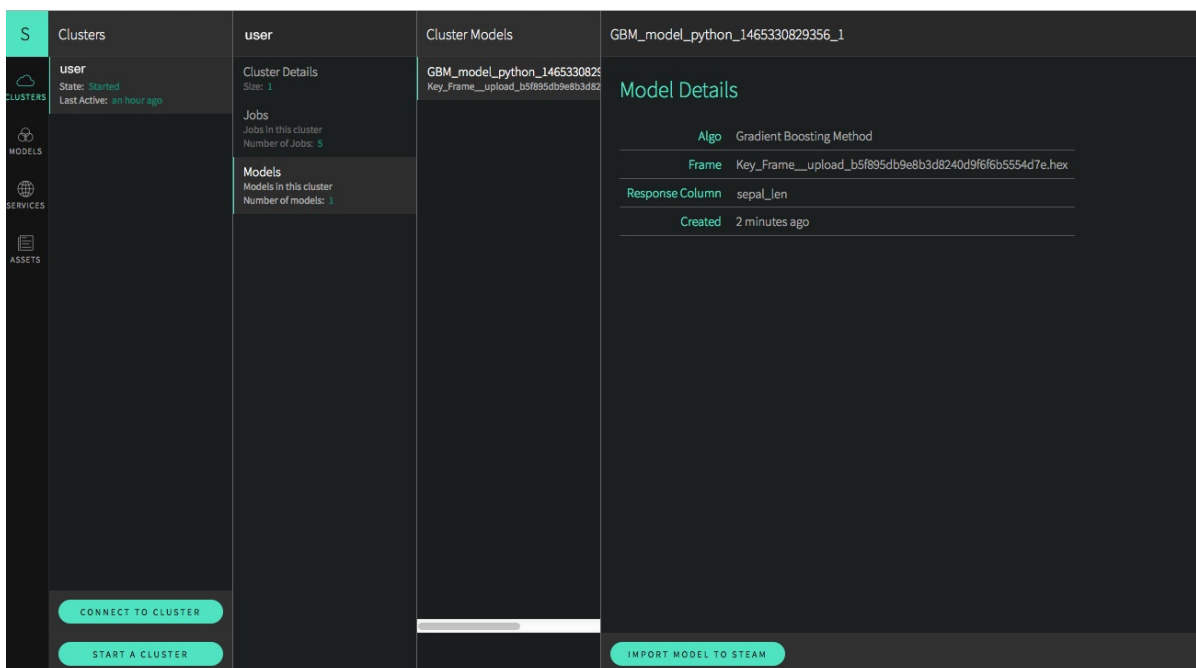
   i. Optionally view the model details.

```
>>> gbm_regressor
```

Once created, the model will be visible in the Steam UI.

Deploying a Model in Steam

1. In the Steam UI, Select **Cluster** > **Models**. Select a model from your demo, and then click **Import Model to Steam**. This pulls the model into Steam. Once imported, the model can then be deployed to the scoring service.

2. Select the **Models** tab in the left pane. You should see the model that you just imported. Select this model, and then click **Deploy This Model** to create scoring services for the model.

```
![](images/deploy_model.png)
```

1. Specify the port number for the scoring service (defaults to 8000), then click **Deploy**.

2. Select the **Services** tab in the left pane.

3. Select a service (in this case, the model you just deployed), and click the link in the **Endpoint** field to reach the scoring service.
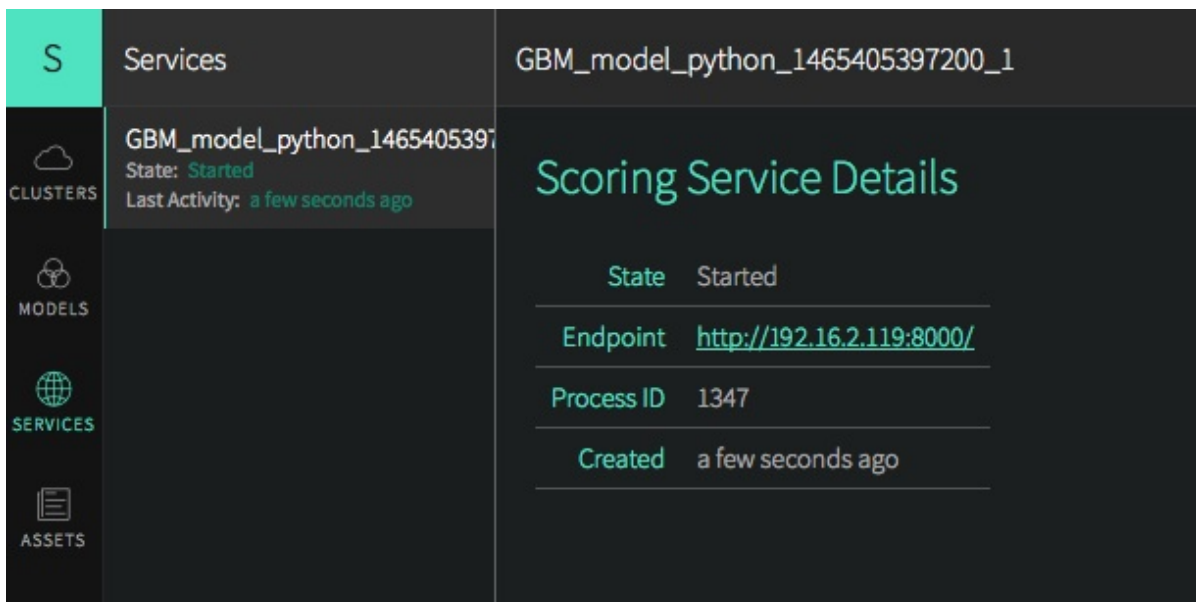
4. Make predictions using one of the following methods:

   o Specify input values based on column data from the original dataset

     OR

   o Enter a query string using the format `field1=value1&field2=value2` (for example, `sepal_width=3&petal_len=5`)

Use the **Clear** button to clear all entries and begin a new prediction.

You can view additional statistics about the scoring service by clicking the **More Stats** button.

Refer to the H2O Scoring Service Builder document for more information.

Stopping the Steam Database

When you are finished using Steam, press Ctrl+C in each of the Steam, Compilation Service, and postgres terminal windows to stop the services end your session.

What's Next?

Now that you have completed your first demo, you are ready to begin creating models using your own data. Additional users can then be give access to this data based on the user's role and workgroup.

H2O Scoring Service Builder

The H2O Scoring Service Builder is a stand-alone application that allows you to perform the following through either a web UI or command line:

1. Compile a POJO, and then build a Jar file from a POJO and a gen-model file
2. Compile the POJO, and then build a War file that is a service from a POJO, gen-model. You can then run the war file in Jetty, Tomcat, etc.
3. Build a War file with a POJO predictor and Python pre-preprocessing

Notes

All code is compiled to Java 1.6 to make it useable with rJava.

If you use an older version of Java, you will need to use an older jetty-runner. Jetty 9.3 requires Java 1.8. Jetty 9.0-9.2 requires Java 1.7. Jetty 8 requires Java 1.6.

Testing has been done on Java 1.6-1.8. For Java 1.8, you can use all Jetty runners. For Java 1.7, you can use all except the 9.3 version. For Java 1.6, you need Jetty 8.

Build and Run the Builder Service

Perform the following steps to build the H2O Scoring Service Builder:

1. In a terminal window, navigate to the **steamY/scoring-service-builder** folder.

2. Run `./gradlew build` to build the service.

3. You will see a **BUILD SUCCESSFUL** message upon completion. Run `./gradlew jettyRunWar` to run the builder service.

4. Open a browser and navigate to localhost:55000 to begin using the H2O Scoring Service Builder web UI.
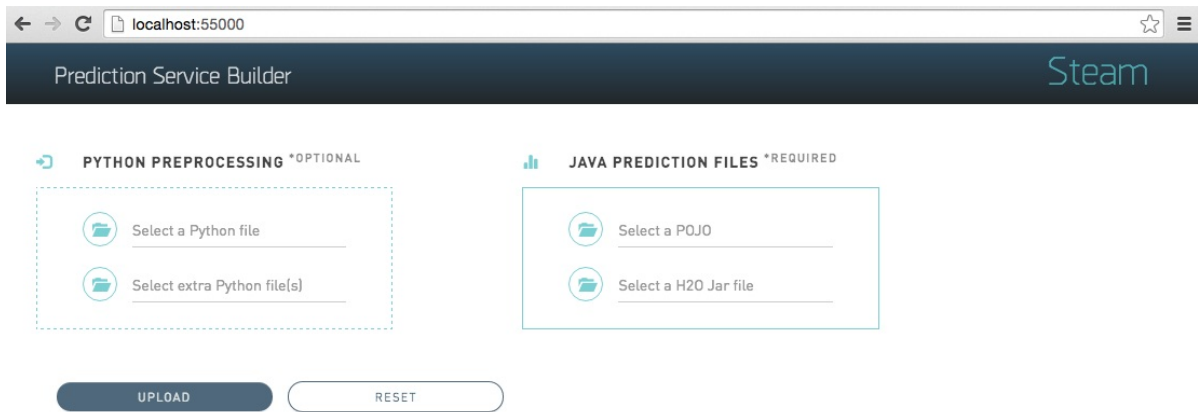
Running an Example War File

Using the Web UI

When the Builder Service is running, you can make a War file using the Web UI.

The following screenshot shows how to make a War file using a POJO file and a Jar file, which are specified in the **JAVA PREDICTION FILES SECTION**. An example model and an H2O Jar file are included in the **steamY/scoring-service-builder/examples/pojo-server** folder.
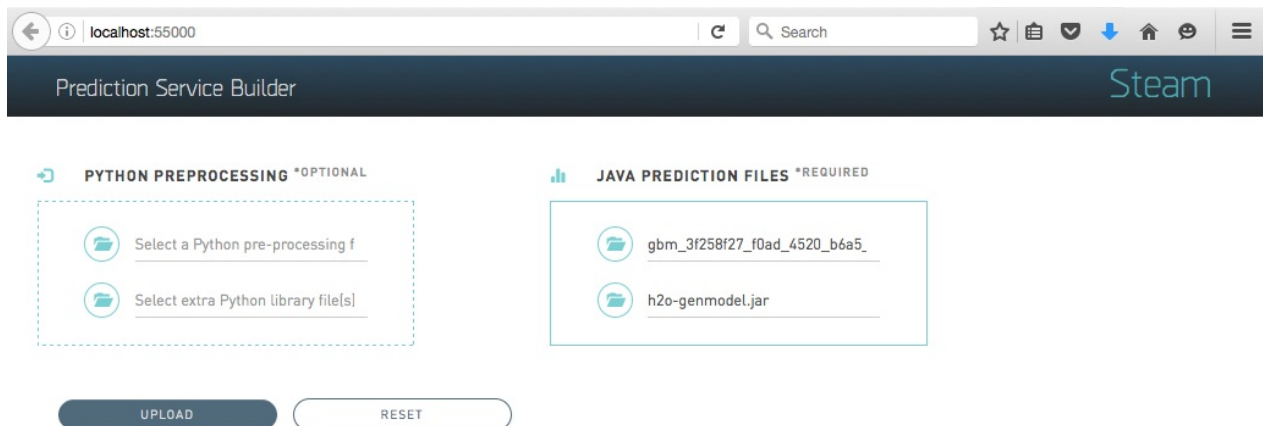
Click **Upload** to upload the files to the Prediction Service Builder and create a War file.

> *Note*: You can also build a War file using a model with an additional pre-processing step. There are two options for this: Python or Java/Scala.
>
> Python. Specify a Python file and additional pre-processing python files and use `makewarpython` . When these files are specified, the prediction file will perform additional pre-processing on the specified model before predicting with the model POJO.
>
> Java/Scala/Jar. Specify a Jar file that contains everyting you need, then the name of the Class that performs the preprocessing. Use the regular `makewar` . When these files are specified, the prediction file will perform additional pre-processing on the specified model before predicting with the model POJO..
>
> For testing, an example python file is available in the **/scoring-service/builder/examples/spam-detection-python** folder. The additional python files are available in the **/scoring-service/builder/examples/spam-detection-python/lib** folder.

Using the CLI

Note that when the Builder Service is running, you can also make a war file using command line arguments. For example:

```
curl -X POST --form pojo=@examples/pojo-server/gbm_3f258f27_f0ad_4520_b6a5_3d2bb4a
9b0ff.java --form jar=@examples/pojo-server/h2o-genmodel.jar localhost:55000/makew
ar > example.war
```

where:

- `gbm_3f258f27_f0ad_4520_b6a5_3d2bb4a9b0ff.java` is the POJO file from H2O
- `h2o-genmodel.jar` is the corresponding Jar file from your version of H2O

The POJO and Jar files are included in the **steamY/scoring-service-builder/examples/pojo-server** folder.

Starting the H2O Prediction Service

You can start the H2O Prediction Service using the CLI, or you can run the example scripts included with this package. Running the scripts will automatically start the H2O Prediction Service.

Using the CLI

Open a terminal window and run the following command to run the scoring service using the gbm_3f258f27_f0ad_4520_b6a5_3d2bb4a9b0ff.war file that was just built:

```
java -jar jetty-runner-9.3.9.M1.jar --port 55001 ~/Downloads/gbm_3f258f27_f0ad
_4520_b6a5_3d2bb4a9b0ff.war
```

This starts the H2O Prediction Service at localhost:55001. You can use this web service at http://localhost:55001.

Using example.sh

This distribution includes scripts that can be used to quickly get your prediction service up and running.

Open a terminal window and run the following commands from the **steamY/steam-scoring-service** folder.

```
# Run an example using the airline dataset
cd examples/pojo-server
sh example.sh
sh run-example.sh
```

or

```
# Run an example using the spam-detection dataset
cd examples/spam-detection-python
sh example-python.sh
sh run-example-python.sh
```

These example scripts generate a War file and then start the prediction service at http://localhost:55001 using a subset of either the airline or spam-detection dataset.

Note: If you run the example-python.sh and run-example-python.sh files, be sure to install textblob (`pip install textblob`) so that the python example runs properly.

Making Predictions

Using the Web UI

The Prediction Service includes a list of the model input parameters that you can specify when making a prediction. The parameters are based on the column headers from the dataset that was used to build the model.

Specify a set of prediction values, or enter an input values query string, then click **Predict** to view the prediction.

Using the CLI

GET

You can send a GET request with the input variables as the query string.

```
curl "localhost:55001/predict?Dest=JFK"
```

This returns a JSON result:

```
{"labelIndex":1,"label":"Y","classProbabilities":[0.026513747179178093,0.973486252
8208219]}
```

The predictor has two classes. "Y" was predicted with a probability of 97%.

POST

JSON can be sent using a POST request.

```
curl -X POST --data '{Dest: JFK}' localhost:55001/predict
```

This returns a JSON result:

```
{"labelIndex":1,"label":"Y","classProbabilities":[0.026513747179178093,0.973486252
8208219]}
```

Batch POST

You can also send multiple JSON inputs at the same time as a batch. Each JSON input has to be on a separate line. If the file `jsonlines.txt` contains

```
{"Dest":"JFK"}
{"Dest":"SFO"}
{"Dest":"JFK"}
```

The command

```
curl -X POST --data-binary @jsonlines.txt  http://localhost:55001/predict
```

then returns

```
{"labelIndex":1,"label":"Y","classProbabilities":[0.026513747179178093,0.973486252
8208219]}
{"labelIndex":1,"label":"Y","classProbabilities":[0.008905417583984554,0.991094582
4160154]}
{"labelIndex":1,"label":"Y","classProbabilities":[0.026513747179178093,0.973486252
8208219]}
```

H2O Prediction Run-Time Stats

Prediction statistics are provided in the web page for the predictor and as a web service in JSON at

```
http://localhost:55001/stats
```

- When the service was started in UTC and its uptime in days
- When the last prediction was run in UTC and how long ago that was in days
- How long time the last prediction took in milliseconds
- Total and average prediction time, with and without skipping the first 5 predictions (warmup)

- There are separate sections for

    - pure predictions using the pojo
    - GET and POST for POJOs
    - GET and POST for POJOs with Python preprocessing
- GET and POST statistics include the time from when the service recieves the input until it returns the result

Click the **More Stats** button to view more Prediction Service statistics.

CLI Command Reference Appendix

- `create identity`
- `create role`
- `create workgroup`
- `deactivate identity`
- `delete cluster`
- `delete engine`
- `delete model`
- `delete role`
- `delete service`
- `delete workgroup`
- `deploy engine`
- `get cluster`
- `get clusters`
- `get engine`
- `get engines`
- `get entities`
- `get history`
- `get identities`
- `get identity`
- `get model`
- `get models`
- `get permissions`
- `get role`
- `get roles`
- `get service`
- `get services`
- `get workgroup`
- `get workgroups`
- `import model`
- `link identity`
- `link role`
- `login`
- `register cluster`
- `reset`

- `start cluster`
- `stop cluster`
- `stop service`
- `unlink identity`
- `unregister cluster`
- `update role`
- `update workgroup`

`create identity`

Description

Creates a new user.

Usage

```
./steam create identity [username] [password]
```

Parameters

- `[username]` : Enter a unique string for the new user name
- `[password]` : Enter a string for the new user's password

Example

The following example creates a new user with a username of "minky" and a password of "m1n5kypassword".

```
./steam create identity minsky m1n5kypassword
Created user minsky ID: 2
```

`create role`

Description

Creates a new role.

Usage

```
./steam create role [rolename] --desc="[description]"
```

Parameters

- `[rolename]` : Enter a unique string for the new role
- `--desc="[description]"` : Optionally enter a string that describes the new role

Example

The following example creates an engineer role.

```
./steam create role engineer --desc="a default engineer role"
Created role engineer ID: 2
```

`create workgroup`

Description

Creates a new workgroup.

Usage

```
./steam create workgroup [workgroupname] --desc="[description]"
```

Parameters

- `[workgroupname]` : Enter a unique string for the new workgroup
- `--desc="[description]"` : Optionally enter a string that describes the new workgroup

Example

The following example creates a data preparation workgroup.

```
./steam create workgroup preparation --desc="data prep group"    Created workgroup
 preparation ID: 1
```

`deactivate identity`

Description

Deactivates an identity based on the specified username.

Usage

```
./steam deactivate identity [username]
```

Parameters

- `[username]` : Specify the username of the identity that you want to deactivate.

Example

The following example deactivates user "minsky".

```
./steam deactivate minsky
```

`delete cluster`

Description

Deletes the specified YARN cluster from the database. Note that this command can only be used with YARN clusters (i.e., those started using `start cluster` .) This command will not work with local clusters. In addition, this commmand will only work on cluster that have been stopped using `stop cluster` .

Usage

```
./steam delete cluster [id]
```

Parameters

- `[id]` : Specify the ID of the cluster that you want to delete.

Example

The following example deletes cluster 2.

```
./steam get engines
NAME               ID    AGE
automl-hdp2.2.jar    1    2016-07-14 11:48:42 -0700 PDT
h2o-genmodel.jar     2    2016-07-14 11:49:47 -0700 PDT
./steam delete engine 1
Engine deleted: 1
```

delete engine

Description

Deletes the specified engine from the database.

Usage

```
./steam delete engine [id]
```

Parameters

- `[id]` : Specify the ID of the engine that you want to delete.

Example

The following example retrieves a list of engines currently added to the database. It then specifies to delete that automodel-hdp2.2.jar engine.

```
./steam get engines
NAME               ID    AGE
automl-hdp2.2.jar    1     2016-07-14 11:48:42 -0700 PDT
h2o-genmodel.jar     2     2016-07-14 11:49:47 -0700 PDT
./steam delete engine 1
Engine deleted: 1
```

delete model

Description

Deletes a model from the database based on the model's ID.

Usage

```
./steam delete model [modelId]
```

Parameters

- `[modelId]` : Specify the ID of the model that you want to delete.

Example

The following example deletes model 3 from the database. Note that you can use [`get models`]'(#get models) to retrieve a list of models.

```
./steam delete model 3
```

delete role

Description

Deletes a role from the database based on its ID.

Usage

```
./steam delete role [roleId]
```

Parameters

- `[roleId]` : Specify the ID of the role that you want to delete.

Example

The following example deletes role 3 from the database. Note that you can use [`get roles`]'(#get roles) to retrieve a list of roles. In the case below, this role corresponds to the default data science role.

```
./steam delete role 3
```

delete service

Description

A service represents a successfully deployed model on the Steam Scoring Service. This command deletes a service from the database based on its ID. Note that you must first stop a service before it can be deleted. (See `stop service` .)

Usage

```
./steam delete service [serviceId]
```

Parameters

- `[serviceId]` : Specify the ID of the service that you want to delete. Note that you can use `get services` to retrieve a list of services.

Example

The following example stops and then deletes service 2. This service will no longer be available on the database.

```
./steam stop service 2
./steam delete service 2
```

`delete workgroup`

Description

Deletes a workgroup from the database based on its ID.

Usage

```
./steam delete workgroup [workgroupId]
```

Parameters

- `[workgroupId]` : Specify the ID of the role that you want to delete.

Example

The following example deletes workgroup 3 from the database. Note that you can use [`get workgroups`]'(#get workgroups) to retrieve a list of workgroups.

```
./steam delete workgroup 3
```

`deploy engine`

Description

Deploys an H2O engine. After an engine is successfully deployed, it can be specified when starting a cluster. (See `start cluster` .)

Usage

```
./steam deploy engine [path/to/engine]
```

Parameters

- `[path/to/engine]` : Specify the location of the engine that you want to deploy.

Example

The following specifies to deploy the H2O AutoML engine.

```
./steam deploy engine ../engines/automl-hdp2.2.jar
```

`get cluster`

Description

Retrieves detailed information for a specific cluster based on its ID.

Usage

```
./steam get cluster [clusterId]
```

Parameters

- `[clusterId]` : Specify the ID of the cluster that you want to retrieve

Example

The following example retrieves information for cluster ID 1.

```
./steam get cluster 1
              user
TYPE:            external
STATE:           healthy
H2O VERSION:     3.8.2.8
MEMORY:          3.56 GB
TOTAL CPUS:      8
ID:              1
AGE:             2016-07-15 09:23:16 -0700 PDT
```

get clusters

Description

Retrieves a list of clusters.

Usage

```
./steam get clusters
```

Parameters

None

Example

The following example retrieves a list of clusters that are running H2O and are registered in Steam. (See `register cluster`.)

```
./steam get clusters
NAME        ID    ADDRESS          STATE     TYPE        AGE
user        1     localhost:54321  started   external    2016-07-01 11:45:58 -0
700 PDT
```

get engine

Description

Retrieves information for a specific engine based on its ID.

Usage

```
./steam get engine [engineId]
```

Parameters

- `[engineId]` : Specify the ID of the engine that you want to retrieve

Example

The following example retrieves information about engine 1.

```
./steam get engine 1
    h2o-genmodel.jar
ID:        1
AGE:    2016-07-15 09:44:10 -0700 PDT
```

`get engines`

Description

Retrieves a list of deployed engines.

Usage

```
./steam get engines
```

Parameters

None

Example

The following example retrieves a list of engines that have been deployed. (Refer to `deploy engine` .)

```
./steam get engines
NAME                ID    AGE
h2o-genmodel.jar    1    2016-07-01 13:30:50 -0700 PDT
h2o.jar             2    2016-07-01 13:32:10 -0700 PDT
```

`get entities`

Description

Retrieves a list of supported Steam entity types.

Usage

```
./steam get entities
```

Parameters

None

Example

The following example retrieves a list of the supported Steam entity types.

```
./steam get entities
NAME        ID
role        1
workgroup   2
identity    3
engine        4
cluster       5
project       6
model       7
service       8
```

`get history`

Description

Retrieves recent activity information related to a specific user or for a specific cluster.

Usage

```
./steam get history [identity [identityName] | cluster [clusterId]]
```

Parameters

- `identity [identityName]` : Specifies to retrieve activity information related to a specific user
- `cluster [clusterId]` : Specifies to retrieve a activity information related to a specific cluster

Example

The following example retrieves information for user "bob".

```
./steam get history identity bob
USER    ACTION    DESCRITPION                      TIME
1       link    {"id":"2","name":"preparation","type":"workgroup"}    2016-07-15
09:32:55 -0700 PDT
1       link    {"id":"2","name":"engineer","type":"role"}        2016-07-15 09:3
2:44 -0700 PDT
1       create    {"name":"bob"}                      2016-07-15 09:32:32 -0700
 PDT
```

`get identities`

Description

Retrieves a list of users.

Usage

```
./steam get identities
```

Parameters

None

Example

The following example retrieves a list of users that are available on the database.

```
./steam get identities
NAME        ID    LAST LOGIN                AGE
bob          2    0000-12-31 16:00:00 -0800 PST    2016-07-15 09:32:32 -0700 PDT
jim          3    0000-12-31 16:00:00 -0800 PST    2016-07-15 09:32:38 -0700 PDT
superuser   1    0000-12-31 16:00:00 -0800 PST    2016-07-15 09:21:58 -0700 PDT
```

`get identity`

Description

Retrieve information about a specific user.

Usage

```
./steam get identity [identityId]
```

Parameters

- `[identityId]` : Specify the ID of the user you want to retrieve

Example

The following example retrieves information about user Jim.

```
./steam get identity jim
            jim
STATUS:        Active
LAST LOGIN:    0000-12-31 16:00:00 -0800 PST
ID:        3
AGE:          2016-07-15 09:32:38 -0700 PDT


WORKGROUP     DESCRIPTION
production    production group


ROLE          DESCRIPTION
datascience    a default data scientist role


PERMISSIONS
Manage models
View clusters
Manage projects
```

`get model`

Description

Retrieves detailed information for a specific model.

Usage

```
./steam get model [modelId]
```

Parameters

- `[modelId]` : Specify the ID of the model that you want to retrieve

Example

The following example retrieves information for model 2.

```
./steam get model 2
```

get models

Description

Retrieves a list of models.

Usage

```
./steam get models
```

Parameters

None

Example

The following example retrieves a list of models that are available on the database.

```
./steam get models
```

get permissions

Description

Retrieves a list of permissions available in Steam along with the corresponding code. These permissions are currently hard coded into Steam.

Usage

```
./steam get permissions
```

Parameters

None

Example

The following example retrieves a list of Steam permissions.

```
./steam get permissions
ID    DESCRIPTION        CODE
9    Manage clusters      ManageCluster
7    Manage engines       ManageEngine
5    Manage identities   ManageIdentity
13   Manage models        ManageModel
11   Manage projects       ManageProject
1    Manage roles        ManageRole
15    Manage services       ManageService
3    Manage workgroups   ManageWorkgroup
10    View clusters        ViewCluster
8    View engines       ViewEngine
6    View identities      ViewIdentity
14    View models        ViewModel
12    View projects       ViewProject
2    View roles       ViewRole
16    View services       ViewService
4    View workgroups       ViewWorkgroup
```

`get role`

Description

Retrieves detailed information for a specific role based on its name.

Usage

```
./steam get role [roleName]
```

Parameters

- `[roleName]` : Specify the name of the role that you want to retrieve

Example

The following example retrieves information about the datascience role.

```
./steam get role datascience
            datascience
DESCRIPTION:    a default data scientist role
ID:        3
AGE:    2016-07-15 09:32:10 -0700 PDT


IDENTITES: 1
NAME      STATUS     LAST LOGIN
jim        Active      0000-12-31 16:00:00 -0800 PST


PERMISSIONS
Manage models
Manage projects
View clusters
```

`get roles`

Description

Retrieves a list of roles.

Usage

```
./steam get roles
```

Parameters

None

Example

The following example retrieves a list of roles that are available on the database.

```
./steam get roles
NAME          ID    DESCRIPTION              AGE
Superuser     1     Superuser                2016-07-14 09:25:30 -0700 PDT
datascience   3    a default data scientist role    2016-07-14 15:39:03 -0700 PDT
engineer     2    a default engineer role        2016-07-14 15:38:10 -0700 PDT
```

`get service`

Description

A service represents a successfully deployed model on the Steam Scoring Service. This command retrieves detailed information about a specific service based on its ID.

Usage

```
./steam get service [serviceId]
```

Parameters

- `[serviceId]` : Specify the ID of the service that you want to retrieve

Example

The following example retrieve information about service 2.

```
./steam get service 2
```

`get services`

Description

A service represents a successfully deployed model on the Steam Scoring Service. This command retrieves a list of services available on the database.

Usage

```
./steam get services
```

Parameters

None

Example

The following example retrieves a list of services that are available on the database.

```
./steam get services
```

`get workgroup`

Description

Retrieves information for a specific workgroup based on its name.

Usage

```
./steam get workgroup [workgroupName]
```

Parameters

- `[workgroupName]` : Specify the name of the workgroup that you want to retrieve

Example

The following example retrieves information about the production workgroup

```
./steam get workgroup production
              production
DESCRIPTION:    production group
ID:        3
AGE:    2016-07-15 09:32:27 -0700 PDT


IDENTITIES: 1
NAME     STATUS     LAST LOGIN
jim        Active      0000-12-31 16:00:00 -0800 PST
```

`get workgroups`

Description

Retrieves a list of workgroups currently available on the database.

Usage

```
./steam get workgroups --identity=[identityName]
```

Parameters

- `--identity=[identityName]` : Optionally specify to view all workgroups associated with a specific user name

Example

The following example retrieves a list of workgroups that are available on the database.

```
./steam get workgroups
NAME          ID    DESCRIPTION          AGE
preparation    2     data prep group          2016-07-15 09:32:21 -0700 PDT
production     3     production group     2016-07-15 09:32:27 -0700 PDT
```

`import model`

Description

Imports a model from H2O based on its ID.

Usage

```
./steam import model [clusterId] [modelName]
```

Parameters

- `[clusterId`]: Specify the H2O cluster that contains the model you want to import
- `[modelName]` : Specify the name of the that you want to import into steam.

Example

The following example specifies to import the GBM_model_python_1468599779202_1 model from Cluster 1.

```
./steam import model 1 GBM_model_python_1468599779202_1
```

`link identity`

Description

Links a user to a specific role or workgroup.

Usage

```
./steam link identity [identityName] [role [roleId] | workgroup [workgroupId]]
```

Parameters

- `[identityName]` : Specify the user that will be linked to a role or workgroup.
- `role [roleId]` : Specify the role that the user will be linked to.
- `workgroup [workgroupId]` : Specify the workgroup that the the user will be linked to.

Example

The following example links user Jim to datascience role and then to the production workgroup.

```
./steam link identity jim role datascience
./steam link identity jim workgroup production
```

`link role`

Description

Links a role to a certain set of permissions.

Usage

```
./steam link role [roleId] [permissionId1 permissionId2 ...]
```

Parameters

- `[roleId]` : Specify the role that the user will be linked to.
- `[permissionId]` : Specify a single permission or a list of permissions to assign to this role.

Example

The following example links the datascience role to the ManageProject, ManageModel, and ViewCluster permissions.

```
./steam link role datascience ManageProject ManageModel ViewCluster
```

`login`

Description

Logs a user in to Steam

Usage

```
./steam login [address:port] --username=[userName] --password=[password]
```

Parameters

- `[address:port]` : Specify the address and port of the Steam server.
- `--username=[userName]` : Specify the username.
- `--password=[password]` : Specify the user's password.

Example

The following example logs user Bob into a Steam instance running on localhost:9000.

```
./steam login localhost:9000 --username=bob --password=bobSpassword
Login credentials saved for server localhost:9000
```

`register cluster`

Description

Registers a cluster that is currently running H2O (typically a local cluster). Once registered, the cluster can be used to perform machine learning tasks through Python, R, and Flow. The cluster will also be visible in the Steam web UI.

Note that clusters that are started using this command can be stopped from within the web UI or using `unregister cluster`. You will receive an error if you attemt to stop registered clusters using the `stop cluster` command.

Usage

```
./steam register cluster [address]
```

Parameters

- `[address]` : Specify the IP address and port of the cluster that you want to register.

Example

The following example registers Steam on localhost:54323. Note that this will only be successful if H2O is already running on this cluster.

```
./steam register cluster localhost:54323
Successfully connected to cluster 2 at address localhost:54323
```

`reset`

Description

Resets the current Steam cluster instance. This removes the current authentication from Steam. You will have to re-authenticate in order to continue to use Steam.

Usage

```
./steam reset
```

Parameters

None

Examples

The following example resets the current Steam instance.

```
./steam reset
Configuration reset successfully. Use 'steam login <server-address>' to re-authent
icate to Steam
```

`start cluster`

Description

After you have deployed engine, you can use this command to start a new cluster through YARN using a specified engine. Note that this command is only valid when starting Steam on a YARN cluster. To start Steam on a local cluster, use `register cluster` instead.

Usage

```
./steam start cluster [id] [engineId] --size=[numNodes] --memory=[string]
```

Parameters

- `[id]` : Enter an ID for this new cluster.
- `[engineId]` : Specify the ID of the engine that this cluster will use. If necessary, use `get engines` to retrieve a list of all available engines.
- `--size=[numNodes]` : Specify an integer for the number of nodes in this cluster.
- `--memory=[string]` : Enter a string specifying the amount of memory available to Steam in each node (for example, "1024m", "2g", etc.)

Example

The following example retrieves a list of engines, then starts a cluster through YARN using one from the list. The cluster is configured with 2 nodes that are 2 gigabytes each.

```
./steam get engines
NAME                ID    AGE
h2o-genmodel.jar    1     2016-07-01 13:30:50 -0700 PDT
h2o.jar             2     2016-07-01 13:32:10 -0700 PDT
./steam start cluster 9 1 --size=2 --memory=2g
```

stop cluster

Description

Stops a YARN cluster that was started through the CLI or web UI. (See `start cluster` .) Note that you will receive an error if you attempt to stop a cluster that was started using `register cluster` .

Usage

```
./steam stop cluster [id]
```

Parameters

- `[id]` : Specify the ID of the cluster that you want to stop. If necessary, use `get clusters` to retrieve a list of clusters.

Example

The following example stops a cluster that has an ID of 9.

```
./steam stop cluster 9
```

57

`stop service`

Description

A service represents a successfully deployed model on the Steam Scoring Service. Use this command to stop a service.

Usage

```
./steam stop service [serviceId]
```

Parameters

- `[serviceId]` : Specify the ID of the scoring service that you want to stop. If necessary, use `get services` to retrieve a list of running services.

Example

The following example stops a service that has an ID of 2.

```
./steam stop service 2
```

`unlink identity`

Description

Removes a user's permissions from a specific role or workgroup.

Usage

```
./steam unlink identity [identityName] [role [roleId] | workgroup [workgroupId]]
```

Parameters

- `[identityName]` : Specify the user that will be unlinked from a role or workgroup

- `role [roleId]` : Specify the role that the user will be unlinked from
- `workgroup [workgroupId]` : Specify the workgroup that the the user will be unlinked from

Example

The following example removes user Jim from the datascience role and then from the production workgroup.

```
./steam unlink identity jim role datascience
./steam unlink identity jim workgroup production
```

unregister cluster

Description

Stops a cluster that was registered through the CLI or the web UI. (See `register cluster` .) Note that this does not delete the cluster. Also note that you will receive an error if you attempt to unregister a cluster that was started using `start cluster` .

Usage

```
./steam unregister cluster [id]
```

Parameters

- `[id]` : Specify the ID of the cluster that you want to stop. If necessary, use `get clusters` to retrieve a list of clusters.

Example

The following example stops a cluster that has an ID of 9.

```
./steam unregister cluster 2
Successfully unregisted cluster %d 2
```

update role

Description

Edits the description and/or name of an existing role. When a role is edited, the edit will automatically propagate to all identities that are associated with this role.

Usage

```
./steam update role [rolename] --desc="[description]" --name="[newRoleName]
```

Parameters

- `[rolename]` : Enter the role name that you want to edit
- `desc="[description]"` : Optionally enter a string that describes the new role
- `name="[newRoleName]"` : Enter a unique string for the new role name

Example

The following example changes the name of the engineer role to be "science engineer".

```
./steam update role engineer --desc="A better engineer" --name="science engineer"
Successfully updated role: engineer
```

`create workgroup`

Description

Edits the description and/or name of an existing workgroup. When a workgroup is edited, the edit will automatically propagate to all identities that are associated with this workgroup.

Usage

```
./steam update workgroup [workgroupname] --desc="[description]" --name="[newWorkgroupName]
```

Parameters

- `[workgroup]` : Enter the workgroup name that you want to edit
- `desc="[description]"` : Optionally enter a string that describes the new workgroup
- `name="[newWorkgroupName]"` : Enter a unique string for the new workgroup name

Example

The following example changes the name of the production workgroup to be "deploy".

```
./steam update workgroup production --desc="A deploy workgroup" --name="deploy"
Successfully updated workgroup: production
```