Deployment options in H2O

Date:03/04/2014

Executive Summary....3
Deployment methods for H2O models....7
H2O provides APIs to produce and consume models for production deployments. H2O models predict at exceptionally fast speeds. Some of the deployment modes used by our customers are discussed below and can be seen in the figure below: 7
1. Java POJO: 7
2. REST API: 9
Python 9
Java applications 10
3. Other Real time event driven systems 11
4. R 11

Executive Summary

The document below describes the various deployment methods available with H2O. The process starts with data scientists building a model on the H2O cluster through R, Web UI or python. It is an iterative process where the data scientists can run a bunch of algorithms' and find the model with the best accuracy. Once the model is build, the next step is to industrialize the model into application code. H2O provides out of the box functionality that can convert models into java code (POJO) which can be then be easily integrated with application code as shown in the Figure 1

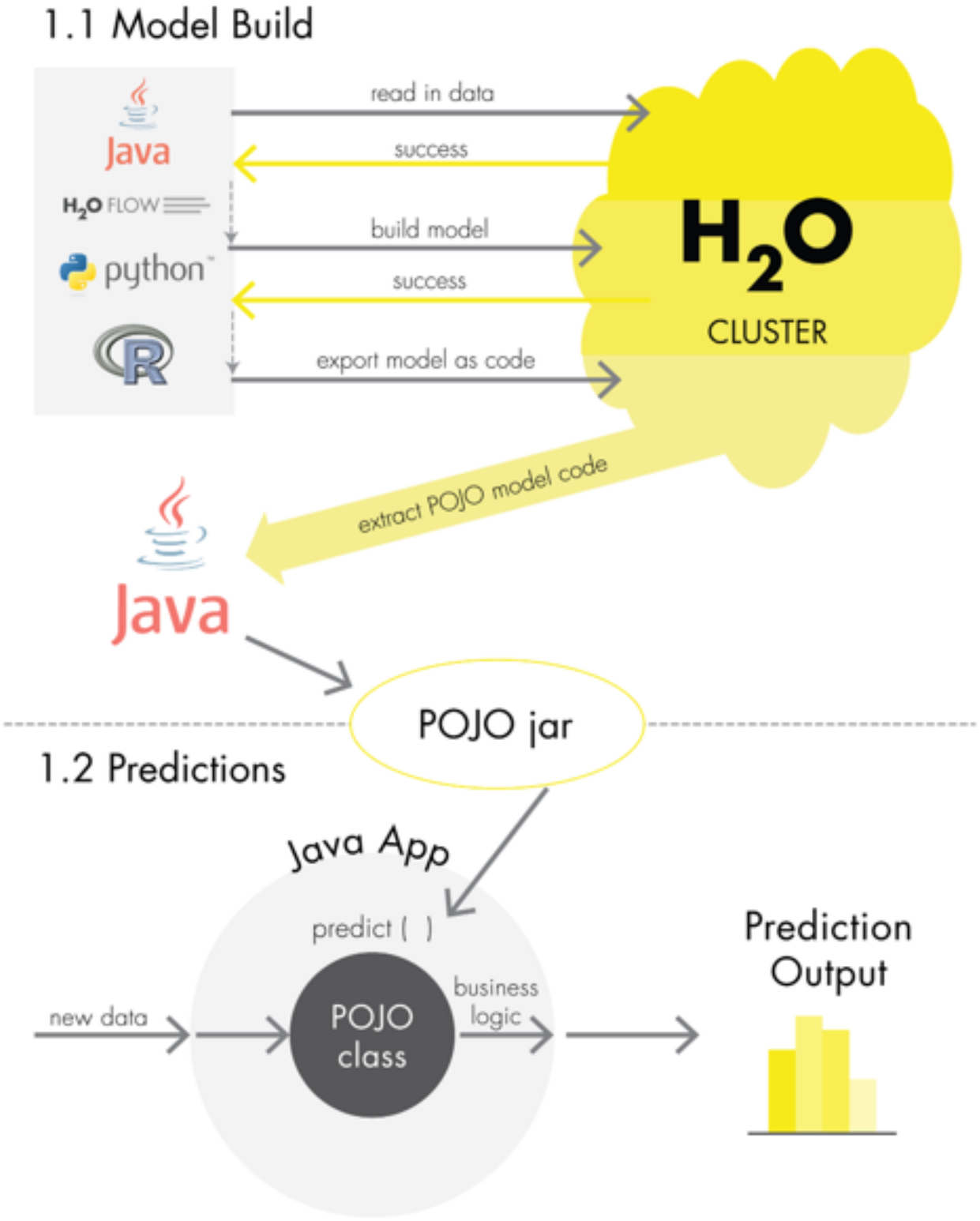

1.1 Model Build
Java
H2O FLOW
python
read in data
success
build model
success
export model as code
H2O
CLUSTER
extract POJO model code
Java
POJO jar
1.2 Predictions
Java App
predict ( )
new data
POJO class
business logic
Prediction Output

Figure 1

Furthermore, H2O exposes a rich set of REST APIs, which can be used to integrate applications with H2O overt REST API calls as shown in Figure 2. Sample code to integrate with java, python and R have been provided.

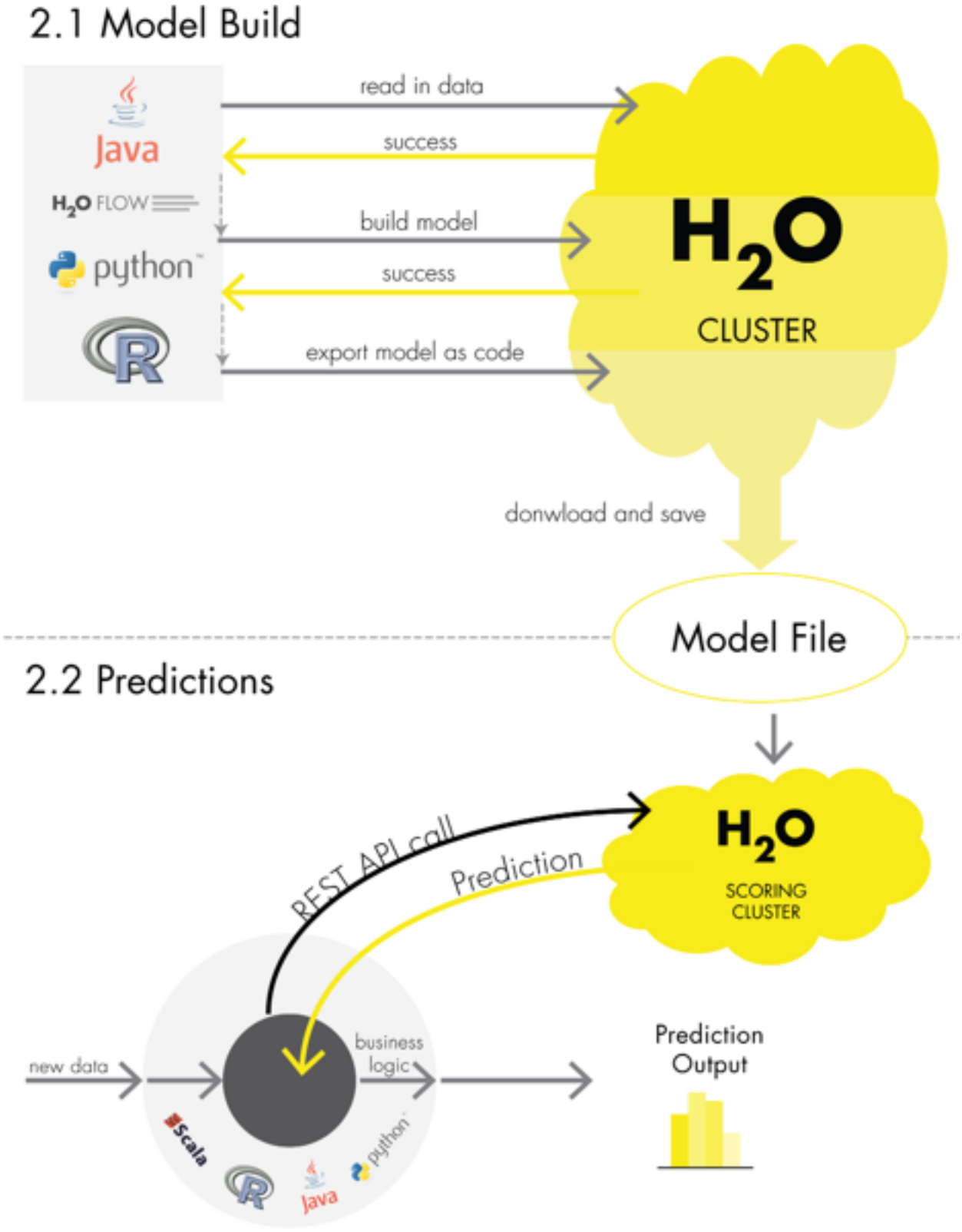

2.1 Model Build
Java
H2O FLOW
python
read in data
success
build model
success
export model as code
H2O
CLUSTER
donwload and save
Model File
2.2 Predictions
REST API call
Prediction
H2O
SCORING
CLUSTER
new data
business
logic
Scala
Java
python
Prediction
Output

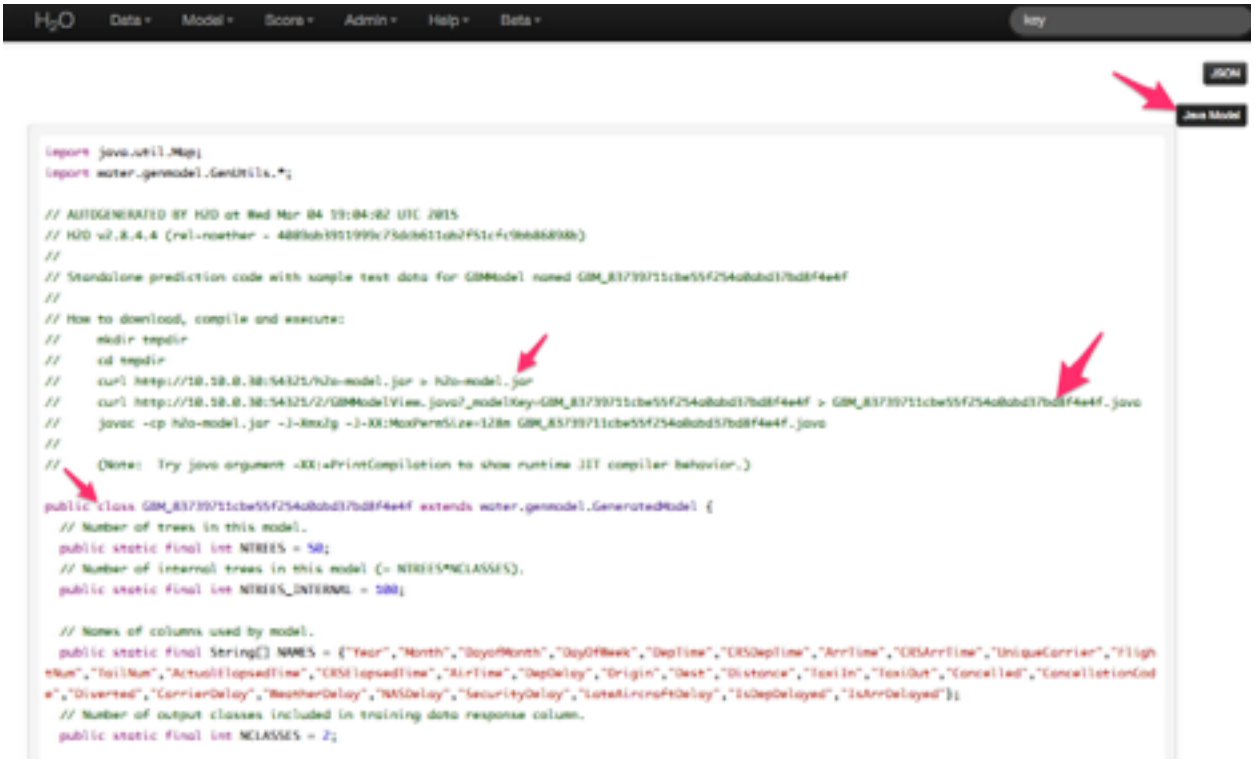
Figure 2

Deployment methods for H2O models

H2O provides APIs to produce and consume models for production deployments. H2O models predict at exceptionally fast speeds. Some of the deployment modes used by our customers are discussed below and can be seen in the figure below:

1. Java POJO:

Once we build the model in H2O, the Web UI interface provides a mechanism to download the model as a java POJO (plain old java object) class (see screen shot below). The POJO is very lightweight and does not depend on any other libraries, not even H2O. As such, the POJO is perfect for embedding into existing applications. This functionality is very unique to H2O and saves a lot of work to convert the models built by a data scientist into java code.



We can also download and compile the java POJO from the command line:

```
// How to download, compile and execute:
mkdir tmpdir
cd tmpdir
curl http://10.10.0.30:54321/h2o-model.jar > h2o-model.jar
curl http://10.10.0.30:54321/2/GBMModelView.java?_modelKey=GBM_83739711cbe55f254a0abd37bd8f4e4f >
GBM_83739711cbe55f254a0abd37bd8f4e4f.java
javac -cp h2o-model.jar -J-Xmx2g -J-XX:MaxPermSize=128m GBM_83739711cbe55f254a0abd37bd8f4e4f.java

// (Note: Try java argument -XX:+PrintCompilation to show runtime JIT compiler behavior.)
```

Deploying java POJO within your java application:

We can embed the java POJO within a java application and then predict/score from within the java application. Sample source code can be found under github at https://github.com/h2oai/h2o/blob/master/R/tests/testdir_javapredict/PredictCSV.java

Example snippet code

```
public static void main(String[] args) throws Exception{
        parseArgs(args);

     water.genmodel.GeneratedModel model;
     model = (water.genmodel.GeneratedModel) Class.forName(modelClassName).newInstance();
…
…
```

```
 for (; j< numInputColumns; j++) row[j] = Double.NaN;
 // Do the prediction.
 //model.predict(row, preds);
 preds = model.predict(row, preds);
…
…
}
```

2. REST API:

H2O exposes a rich rest of REST APIs, which can be used to call the H2O server to make a prediction. The documents for the REST interface can be found at http://docs.h2o.ai/developuser/rest.html.

Python

Our next H2O release (H2O-DEV) will support python API as well. So users comfortable with python can build models and score/predict from python interface. Sample source code python can be found under github at https://github.com/h2oai/h2o-dev/tree/master/h2o-py/demos.

Example code:

```
import sys
import h2o
# Connect to a pre-existing cluster
h2o.init()
df = h2o.import_frame(path="smalldata/logreg/prostate.csv")
df.describe()
# Remove ID from training frame
train = df.drop("ID")
# For VOL & GLEASON, a zero really means "missing"
vol = train['VOL']
vol[vol == 0] = None
gle = train['GLEASON']
gle[gle == 0] = None
# Convert CAPSULE to a logical factor
train['CAPSULE'] = train['CAPSULE'].asfactor()
# See that the data is ready
train.describe()
# Run GBM
my_gbm = h2o.gbm( y=train["CAPSULE"],
                  validation_y=train["CAPSULE"],x=train[1:],validation_x=train[1:],
ntrees=50, learn_rate=0.1)

my_gbm_metrics = my_gbm.model_performance(train)
my_gbm_metrics.show()
```

Java applications

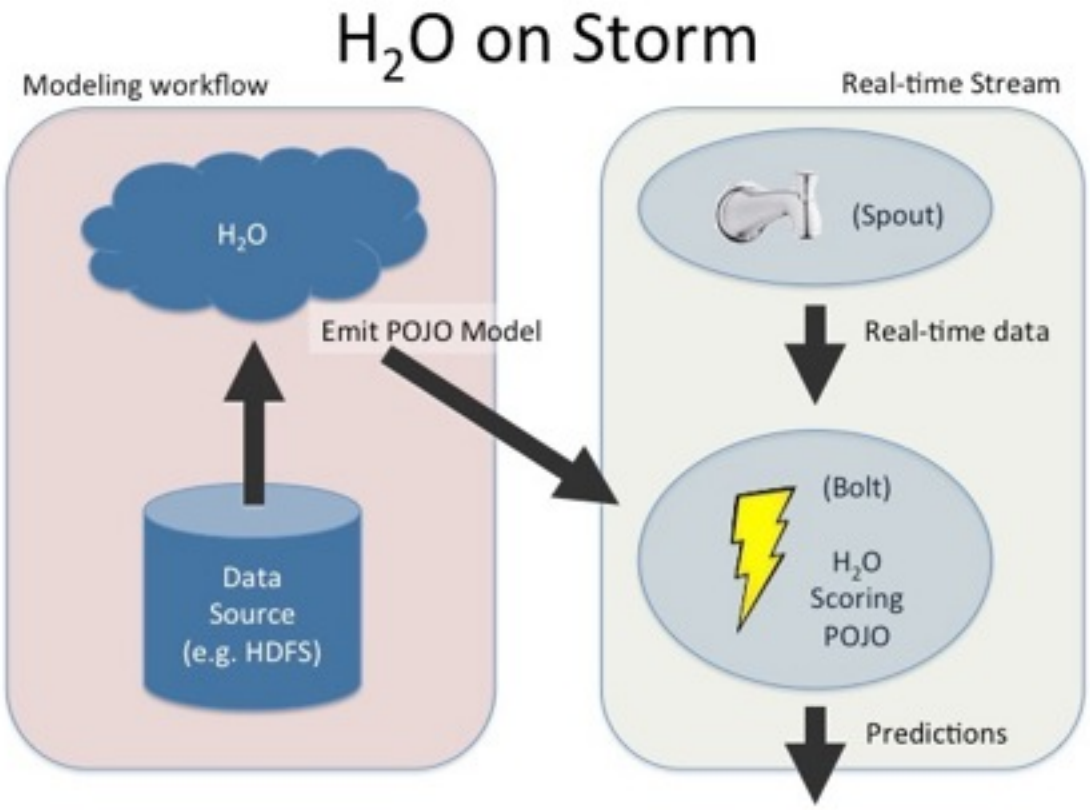
We can write java applications that make REST calls to the backend application. We can also provide an example of using spring application that has an end to end flow of uploading the data file, parsing, scoring and then predicting in real time using a REST call.

Example of building a deep learning model using REST API within a java application.

```
public String BuildDeepLearningModel(String hexkey) {
    String h2oUrlDeepLearningEndPoint = H2O_HOST_URL + H2O_DEEP_LEARNING;
    log.debug("@@@ Calling endpoint {}", h2oUrlDeepLearningEndPoint);

    MultiValueMap<String, String> postParameters =
       new LinkedMultiValueMap<String, String>();

    postParameters.add("training_frame", hexkey);
    postParameters.add("destination_key", H2O_DEEP_LEARNING_DESTINATION_KEY);
    postParameters.add("do_classification", "false");
    postParameters.add("activation", "Tanh");
    postParameters.add("epochs", "10");
    postParameters.add("hidden", "[256]");
    postParameters.add("autoencoder", "true");
    postParameters.add("loss", "MeanSquare");


    try {
        RestTemplate restTemplate = new RestTemplate();
        HttpHeaders headers = new HttpHeaders();
        headers.add("Accept", MediaType.APPLICATION_JSON_VALUE);
        HttpEntity<MultiValueMap<String, String>> request =
          new HttpEntity<MultiValueMap<String, String>>(postParameters, headers);
        ResponseEntity<String> response =
           restTemplate.exchange(h2oUrlDeepLearningEndPoint, HttpMethod.POST, request,
           String.class);
        String responseBody = response.getBody();
        log.debug("@@@ Deep Learning Response json from h2o {}", responseBody);

        JSONObject jsonobject = new JSONObject(responseBody);
        JSONArray jsonarray = (JSONArray) jsonobject.get("jobs");
        JSONObject job = jsonarray.getJSONObject(0); // Always the first element.
        JSONObject key = (JSONObject) job.get("key");

        log.debug("!!!!!! Job name {}", key.get("name"));
        return JobStatus((String) key.get("name"));

    } catch (HttpClientErrorException e) {
        log.debug("client error {}", e.getRootCause().getMessage());
        return null;
    } catch (Exception ex) {
        log.debug("Error occurred in deep learning {}", ex.getMessage());
        return null;
    }
}
```

3. Other Real time event driven systems

An example of real time scoring using POJO using Apache Storm can be found here: http://learn.h2o.ai/content/demos/streaming_data.html



4. R

Data scientists can use R directly to work with H2O to make predictions. The R package uses REST under the covers to interact with the H2O server.

Example R scripts

```
# Create connection to H2O
library(h2o)
h = h2o.init(nthreads = -1)

# Load data
data.train = h2o.importFile(h, "/home/airlines_all.csv")
```

```
# Build model
myY = "IsDepDelayed"
myX = c("Origin", "Dest", "DayofMonth", "Year", "UniqueCarrier", "DayOfWeek",
        "Month", "DepTime", "ArrTime", "Distance")
model = h2o.glm(y = "IsDepDelayed", x = myX, data = data.train, family = "binomial")

# Make predictions
data.test = h2o.importFile(h, "/home/airlines_test.csv")
preds = h2o.predict(model, data.test)
```