Submit a pyspark job through an ActionScript

Steps:

1. Create python script for spark-submit. In the sample below, we have a Random Forest model as a python script by accessing input files from dtap(dtap://TenantStorage/data/sample_libsvm_data.txt) and writing output data back to dtap(dtap://TenantStorage/output_test) location.

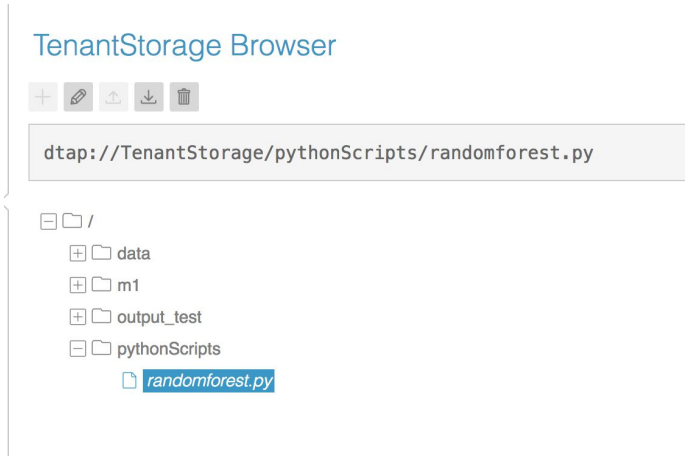
   Important notes:
   Add these three lines in your python script to create SparkContext(sc), if it's not included already.

   from pyspark import SparkContext, SparkConf
   # #Spark Config
   conf = SparkConf().setAppName("sample_app")
   sc = SparkContext(conf=conf)

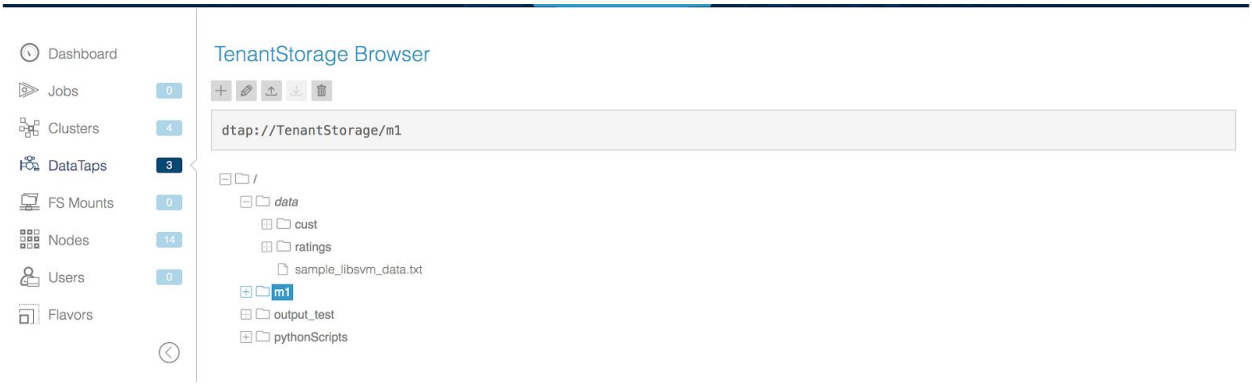
Script name: randomforest.py. Can also be tested from CLI by running *' spark-submit randomforest.py '* from Jupyter node.

```
import sys

from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark import SparkContext, SparkConf

# #Spark Config
conf = SparkConf().setAppName("sample_app")
sc = SparkContext(conf=conf)

# Load and parse the data file into an RDD of LabeledPoint.

data = MLUtils.loadLibSVMFile(sc, 'dtap://TenantStorage/data/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.

model = RandomForest.trainRegressor(trainingData, categoricalFeaturesInfo={},
                                    numTrees=3, featureSubsetStrategy="auto",
                                    impurity='variance', maxDepth=4, maxBins=32)
# Evaluate model on test instances and compute test error

predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testMSE = labelsAndPredictions.map(lambda vp : (vp[0] - vp[1]) * (vp[0] - vp[1])).sum() /\
    float(testData.count())
print('Test Mean Squared Error = ' + str(testMSE))
print('Learned regression forest model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "dtap://TenantStorage/output_test")
sameModel = RandomForestModel.load(sc, "dtap://TenantStorage/output_test")
print(sameModel)
~
~
```

2. If your script is in on your desktop, upload it to dtap location. In this example below, the script is uploaded to dtap://TenantStorage/pythonScripts/randomforest.py



3. Make Sure all the input and output directories defined in python exist in dtap storage. As you can see below, the input file (sample_libsvm_data.txt) used in python script output directory are in dtap.



4. Now let's create an ActionScript "*pyspark_dtap.sh*" to perform spark-submit on the python script we created.
The sample ActionScript below performs the following

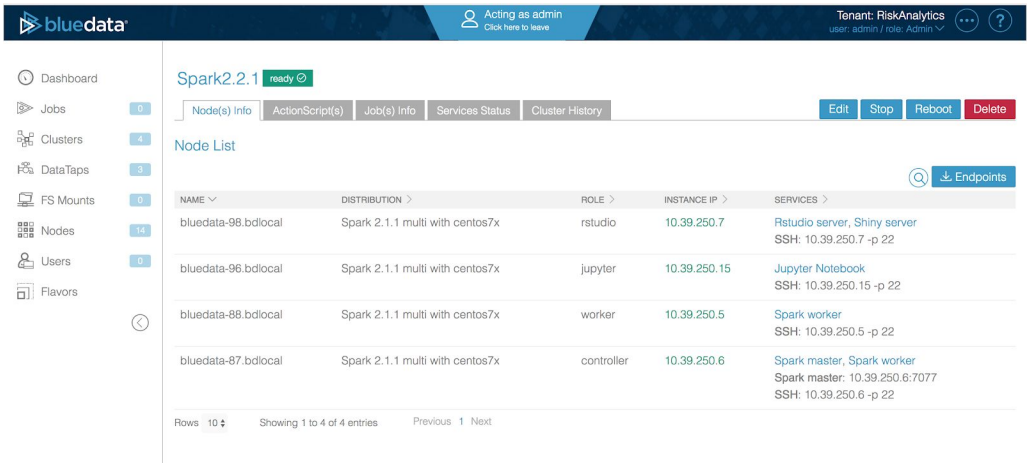
- Copies python script to the node where you are performing spark-submit, under "/tmp" location.

- In this case, script is running on jupyter node, and hence the $node == “jupyter” in the action script. By default, action scripts run on all nodes unless specified. you can replace it with appropriate node where you want to perform spark-submit.
- Makes the script executable.
- Run spark-submit on the python script.

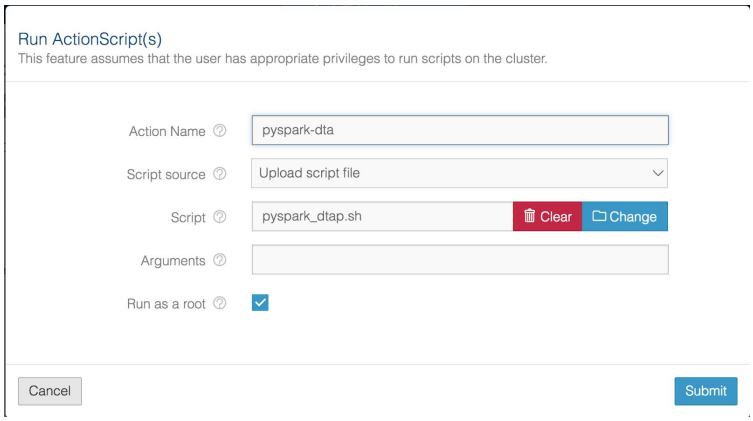
pyspark_dtap.sh

```
#/bin/bash

export node=`bdvcli --get node.role_id`

if [[ $node == "jupyter" ]]; then

    hadoop fs -get dtap://TenantStorage/pythonScripts/randomforest.py /tmp
    chmod a+x /tmp/randomforest.py
    /usr/lib/spark/spark-2.1.1-bin-hadoop2.7/bin/spark-submit /tmp/randomforest.py
fi

```

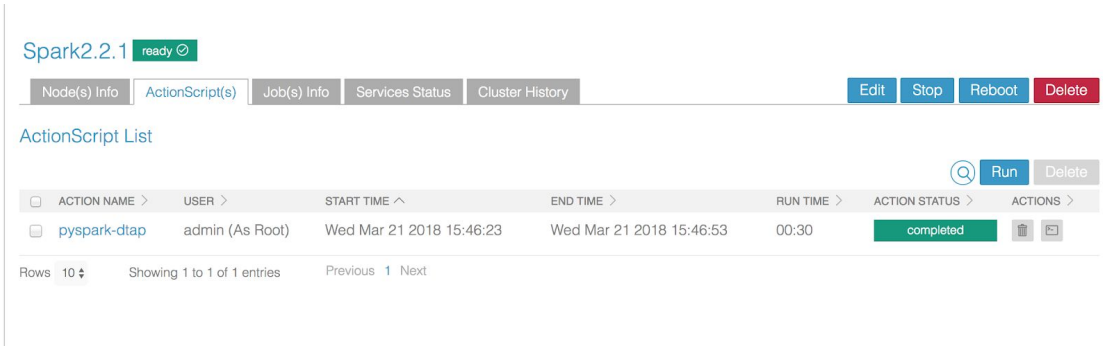
Role definitions are as shown below;



4. To run an action script, create an action as shown below from BlueData cluster detail screen



This submits a spark job to the cluster, you will see the script running with its status



If you click on the logs, under individual actions, You will see the log files as shown below. Here, this script only ran on Jupyter node.