## Image Name: Spark 2.1.1 with Notebooks

| | |
|---|---|
| Location | https://s3.amazonaws.com/bluedata-catalog/solutions/bins/bdcatalog-centos-bluedata-spark211n-1.0.bin |
| DistroId | bluedata/spark211n |
| Version | 1.0 |
| Category (cluster Type) | Spark |
| Software Included | java version "1.8.0_40"<br>Anaconda3-4.0.0 Linux<br>Python 3.5.3 \|Anaconda custom (64-bit)\| running in a virtual environment<br>R version 3.3.3 (2017-03-06)<br>Spark-2.1.1-bin-hadoop2.7<br>Jupyter Notebook installed using conda install<br>Jupyter Toree kernels - Scala, PySpark, SQL (toree-0.2.0.dev1.tar.gz)<br>Zeppelin - zeppelin-0.8.0-SNAPSHOT.tar.gz |
| Notebook access | Jupyter single user hash password - “admin123”<br>Zeppelin default user - “admin” password - “admin123” |
| Systemv Service names and commands | sudo service jupyter-server status (stop, start)<br>sudo service zeppelin-server status(stop, start)<br>sudo service spark-master status (start, stop)<br>sudo service spak-slave status (start, stop) |
| OS | Centos. Works on both Centos and RHEL base machines |

### Sample Code for Testing

### PySpark

```
from pyspark import SparkConf, SparkContext
from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier
import pandas as pd
import numpy as np

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=12000,
                           n_features=10,
                           n_informative=3,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=0,
                           shuffle=False)

# Partition data
def dataPart(X, y, start, stop): return dict(X=X[start:stop, :], y=y[start:stop])
```

```
def train(data):
    X = data['X']
    y = data['y']
    return ExtraTreesClassifier(n_estimators=100,random_state=0).fit(X,y)

# Merge 2 Models
from sklearn.base import copy
def merge(left,right):
    new = copy.deepcopy(left)
    new.estimators_ += right.estimators_
    new.n_estimators = len(new.estimators_)
    return new

data = [dataPart(X, y, 0, 4000), dataPart(X,y,4000,8000), dataPart(X,y,8000,12000)]
forest = sc.parallelize(data).map(train).reduce(merge)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
          axis=0)
indices = np.argsort(importances)[::-1]
# Print the feature ranking
print("Feature ranking:")
for f in range(10):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

## Spark Scala

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.stat.Statistics
import org.apache.spark.rdd.RDD

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5))  // a series
// must have the same number of partitions and cardinality as seriesX
val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If a
// method is not specified, Pearson's method will be used by default.
val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")
println(s"Correlation is: $correlation")
 val data: RDD[Vector] = sc.parallelize(
  Seq(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(5.0, 33.0, 366.0))
)  // note that each Vector is a row and not a column
// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method
// If a method is not specified, Pearson's method will be used by default.
val correlMatrix: Matrix = Statistics.corr(data, "pearson")
println(correlMatrix.toString)
```

## SparkR

```
library(data.table)
dt <- data.table(1:3)
print(dt)
for (i in 1:5) {
  print(i*2)
}
print(1:50)
```