Contents

Oracle Big Data Spatial: Hands-on Lab

Introduction

Welcome to the Hands-on Introduction to the spatial feature of Oracle Big Data Spatial and Graph.

Time to Complete

Perform all parts – 182 Minutes

This lab has six parts:

1. Use Oracle Big Data Spatial and Graph Vector Console (18 mins)
2. Use Oracle Big Data Spatial and Graph Vector Command Line (20 mins)
3. Create Customized Jobs Using the Oracle Big Data Spatial and Graph Vector API (26 mins)
4. Use Oracle Big Data Spatial and Graph Raster Console (26 mins)
5. Use Oracle Big Data Spatial and Graph Raster Command Line (52 mins)
6. Create Customized Jobs Using the Oracle Big Data Spatial and Graph Raster API (35 mins)

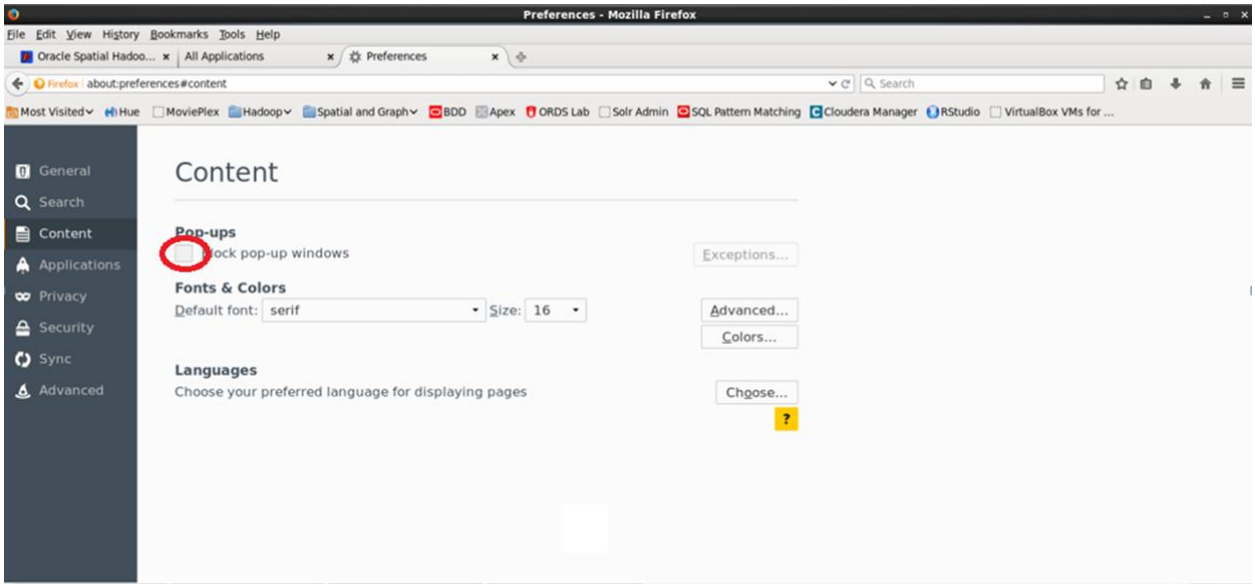
The parts one and four are using the consoles to run and visualize results. Note that you can use the vector console to visualize results created from the command line as well. In the parts two and five you will be able to add more customization to your jobs by running them in the command line. Finally, parts three and six walk you through the creation of Hadoop jobs using Java to generate customized results.

These parts can be completed in any order so if you are particularly interested in one of them, do that one first in case you run out of time.

Let's get started.

1. Install Big Data Lite from http://www.oracle.com/technetwork/database/bigdata-appliance/oracle-bigdatalite-2104726.html
2. Copy Vector-HOL.zip and Raster-HOL.zip to the Big Data Lite Virtual Machine. You can do this following the instructions of the tip **Sharing folders between Big Data Lite and its host** of the Deployment Guide that can be found in the following link http://www.oracle.com/technetwork/database/bigdata-appliance/bigdatalite-quickdeploy-430-2844847.pdf
3. Open the web browser Firefox by clicking the icon below.

4. In Firefox go to Edit/Preferences and uncheck the Block pop-up windows checkbox.



5. Open the terminal.

6. Unzip the file Vector-HOL.zip to the directory /opt/oracle/oracle-spatial-graph/spatial/vector/HOL.

```
unzip Vector-HOL.zip -d /opt/oracle/oracle-spatial-graph/spatial/vector/HOL
```

7. Unzip the file Raster-HOL.zip to the directory /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL.

```
unzip Raster-HOL.zip -d /opt/oracle/oracle-spatial-graph/spatial/raster/
```

8. Start Jetty.

```
/home/oracle/scripts/install-jetty-bdsg.sh

cd /u01/oracle-spatial-graph/spatial/jetty

java -jar start.jar
```

Note: When starting jetty the warning below is expected and can be ignored:

```
2360 [main] ERROR org.apache.hadoop.conf.Configuration  - Failed to set setXIncludeAware(true) for parser oracle.xml.jaxp.JXDocumentBuilderFa
ctory@1dfd5f51:java.lang.UnsupportedOperationException:  setXIncludeAware is not supported on this JAXP implementation or earlier: class orac
le.xml.jaxp.JXDocumentBuilderFactory
java.lang.UnsupportedOperationException:  setXIncludeAware is not supported on this JAXP implementation or earlier: class oracle.xml.jaxp.JXD
ocumentBuilderFactory
        at javax.xml.parsers.DocumentBuilderFactory.setXIncludeAware(DocumentBuilderFactory.java:584)
        at org.apache.hadoop.conf.Configuration.loadResource(Configuration.java:2433)
        at org.apache.hadoop.conf.Configuration.loadResources(Configuration.java:2402)
        at org.apache.hadoop.conf.Configuration.getProps(Configuration.java:2319)
        at org.apache.hadoop.conf.Configuration.get(Configuration.java:895)
        at org.apache.hadoop.conf.Configuration.getTrimmed(Configuration.java:945)
        at org.apache.hadoop.conf.Configuration.getClass(Configuration.java:2106)
        at org.apache.hadoop.fs.FileSystem.getFileSystemClass(FileSystem.java:2578)
        at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:2591)
        at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:91)
        at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:2630)
        at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:2612)
        at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:370)
        at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:169)
        at oracle.spatial.imageserver.HadoopEngine.getFileSystem(HadoopEngine.java:355)
        at oracle.spatial.imageserver.util.conf.ServerConfiguration.init(ServerConfiguration.java:609)
        at oracle.spatial.imageserver.servlets.listeners.Context.contextInitialized(Context.java:48)
```

9. Create what will be our working directory in HDFS:

```
hadoop fs -mkdir /user/oracle/HOL
```

10. Now load the data that we will use during our examples into HDFS:

```
hadoop fs -put /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/data/tweets.json /user/oracle/HOL/tweets.json

hadoop fs -put /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/data/USA_2012Q4_PCB3_PLY.dbf
/user/oracle/HOL/USA_2012Q4_PCB3_PLY.dbf

hadoop fs -put /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/data/USA_2012Q4_PCB3_PLY.shp
/user/oracle/HOL/USA_2012Q4_PCB3_PLY.shp

hadoop fs -put /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/data/USA_2012Q4_PCB3_PLY.shx
/user/oracle/HOL/USA_2012Q4_PCB3_PLY.shx
```

Lab Part 1: Use Oracle Big Data Spatial and Graph Vector Console

Here is the first part of the lab where you will use the Vector Console to:

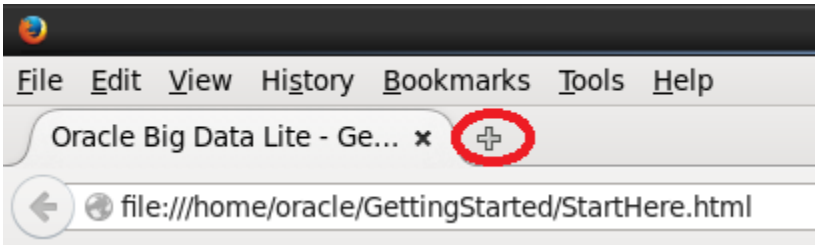
In this lab, the file /user/oracle/HOL/tweets.json will be used. This file is a GeoJSON file with sample tweets. The tweets contain the geometry information, a location text, the number of followers and the number of friends of the person that sent the tweet.

Note: For simplicity our examples won't use the MVSuggest data enrichment service and won't send notification emails after job completions.

Note: The API provides InputFormats and RecordInfoProvider implementation for the common formats GeoJSON and ESRI Shapefiles. It is possible to use any Hadoop provided or customized InputFormat and any customized RecordInfoProvider.

Step 1: Open the web browser Firefox by clicking the icon below.

Note that if you need to open a new tab in Firefox click 

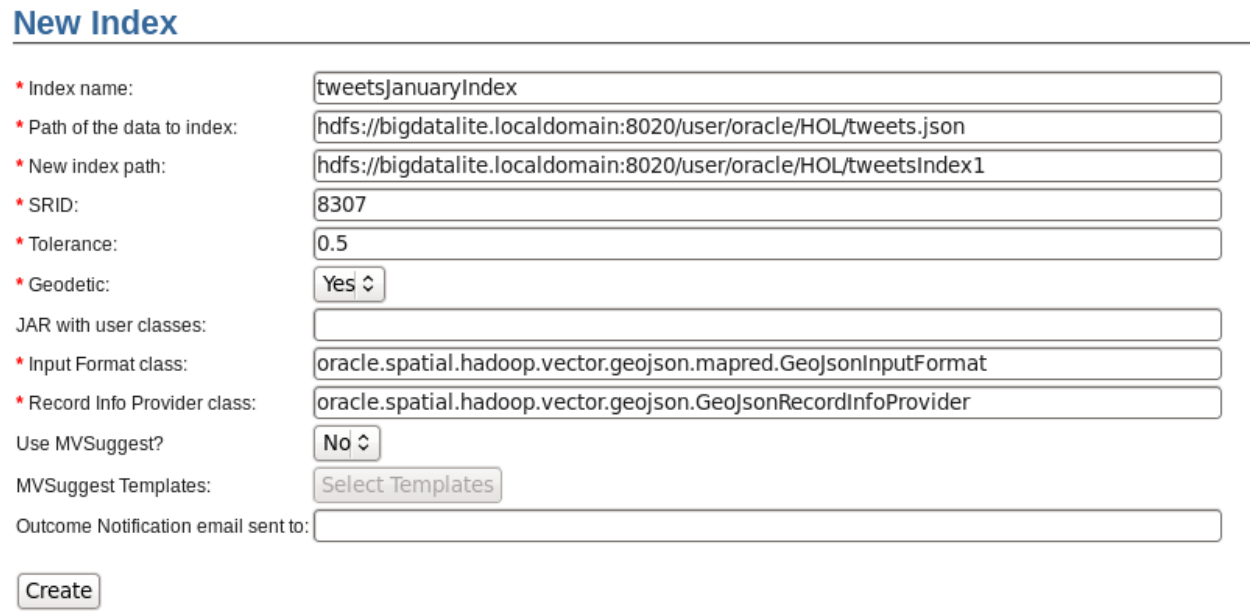


Create Spatial Index

The first task is to create an index on the file /user/oracle/HOL/tweets.json.

1. Open http://localhost:8045/spatialviewer/
2. Click Create Index 

3. Specify all the required details:
    a. The index name: tweetsJanuaryIndex
    b. Path of data to the index: Path of the file(s) to index in HDFS. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/tweets.json*
    c. New index path: This is the job output path. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/tweetsIndex1*
    d. The SRID of the geometries used to build the index: 8307
    e. The tolerance of the geometries used to build the index: The tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors). For this example we set 0.5
    f. If the geometries used to build the index are geodetic or not: Yes
    g. Input Format class: The InputFormat class implementation used to read the input data. For this example we set oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat
    h. Record Info Provider class: The class that provides the spatial information. For this example we set oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider.



4. Click Create.
5. Open http://localhost:8088/cluster/apps in a new Firefox tab and wait until the job is completed successfully (refresh with F5 to see the job updates).

Run a Categorization Job

Before running this example create the sample index performing the task Create Spatial Index. In this task we will categorize the tweets of the file /user/oracle/HOL/tweets.json by Countries and States Provinces so that we know how many tweets have been sent by countries and by Provinces. For example we will know that 736 have been sent from the United States and 47 from California.

1. Open http://localhost:8045/spatialviewer/
2. Click on the section Categorization->Run Job.

3. Select the *With Index* option and select the index *tweetsJanuaryIndex* created in the Task Create Spatial Index.

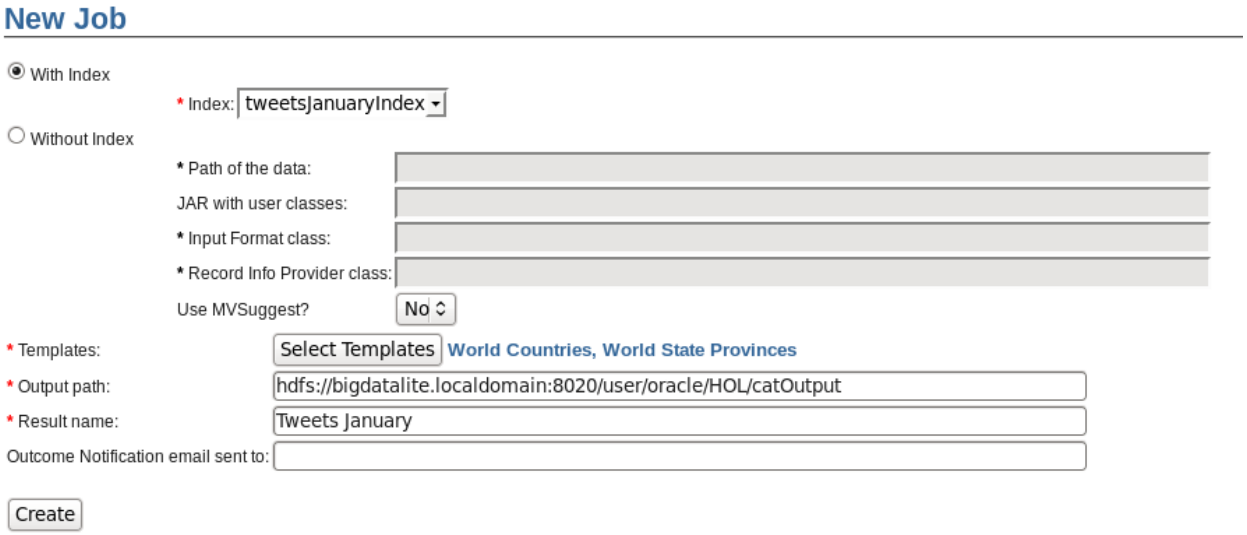
4. Click the Select templates button.

5. Select as hierarchy 1 World Countries and as hierarchy 2 World State Provinces and click Save.

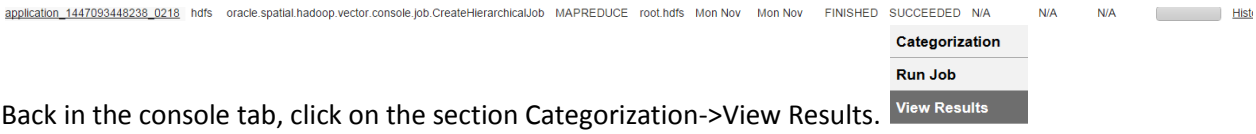
6. Specify the required fields:

a. Output path: The Hadoop job output path. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/catOutput*

b. Result Name: Tweets January

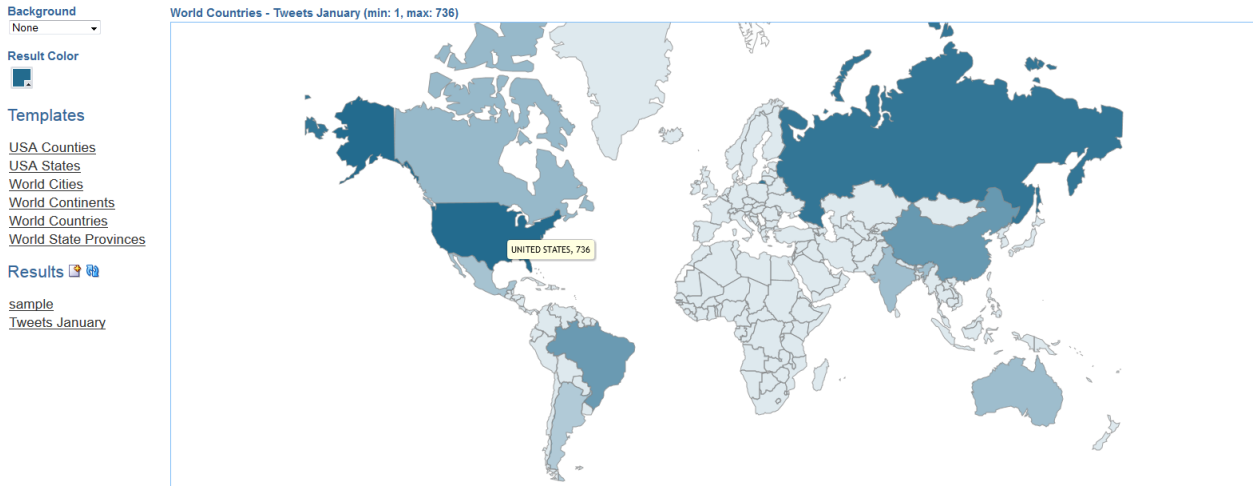


7. Click Create.
8. Open http://localhost:8088/cluster/apps in a new Firefox tab and wait until the job is completed successfully (refresh with F5 to see the job updates).



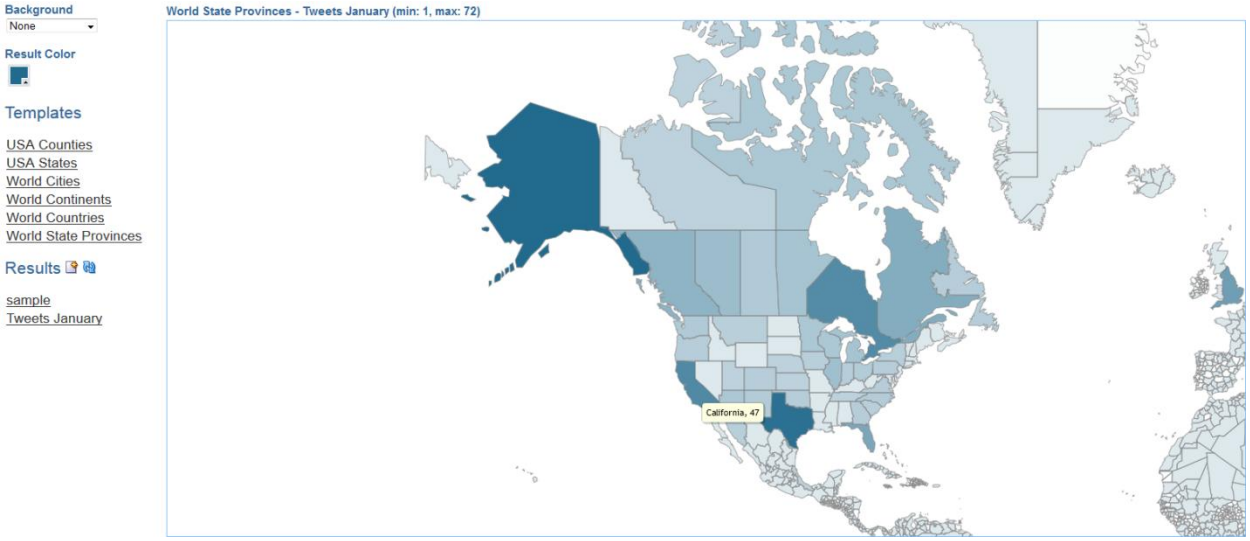
9. Back in the console tab, click on the section Categorization->View Results.

10. Click on the template World Countries.
11. Click on the result Tweets January.



12. See the number of tweets in the United States moving the mouse cursor on it.

13. Click the country United States.

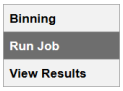


14. The view changes to show the number of tweets by World State Provinces. The focus is on the states of the United States.

Run a Binning Job

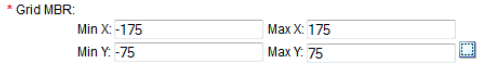
Before to run this example create the sample index performing the Task Create Spatial Index. In this task we will run a binning analysis on the tweets.

1. Open http://localhost:8045/spatialviewer/
2. Click on the section Binning->Run Job.

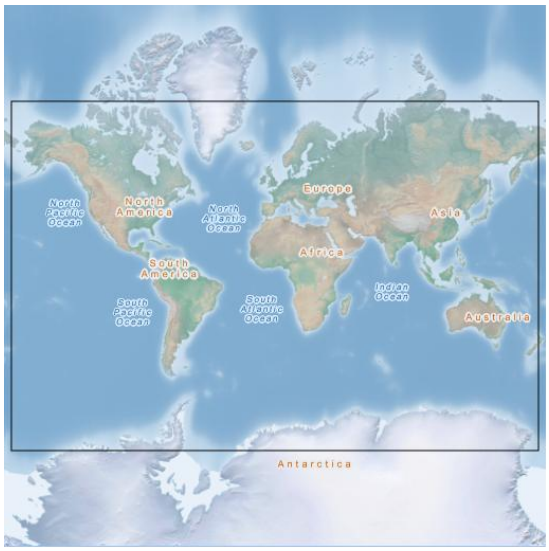


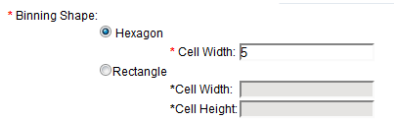
3. Select the *With Index* option and select the index *tweetsJanuaryIndex* created in the task Create Spatial Index.

4. Change the binning grid minimum bounding rectangle (MBR). Set:
   - Min. X to -175
   - Max. X to 175
   - Min. Y to -75
   - Max. Y to 75



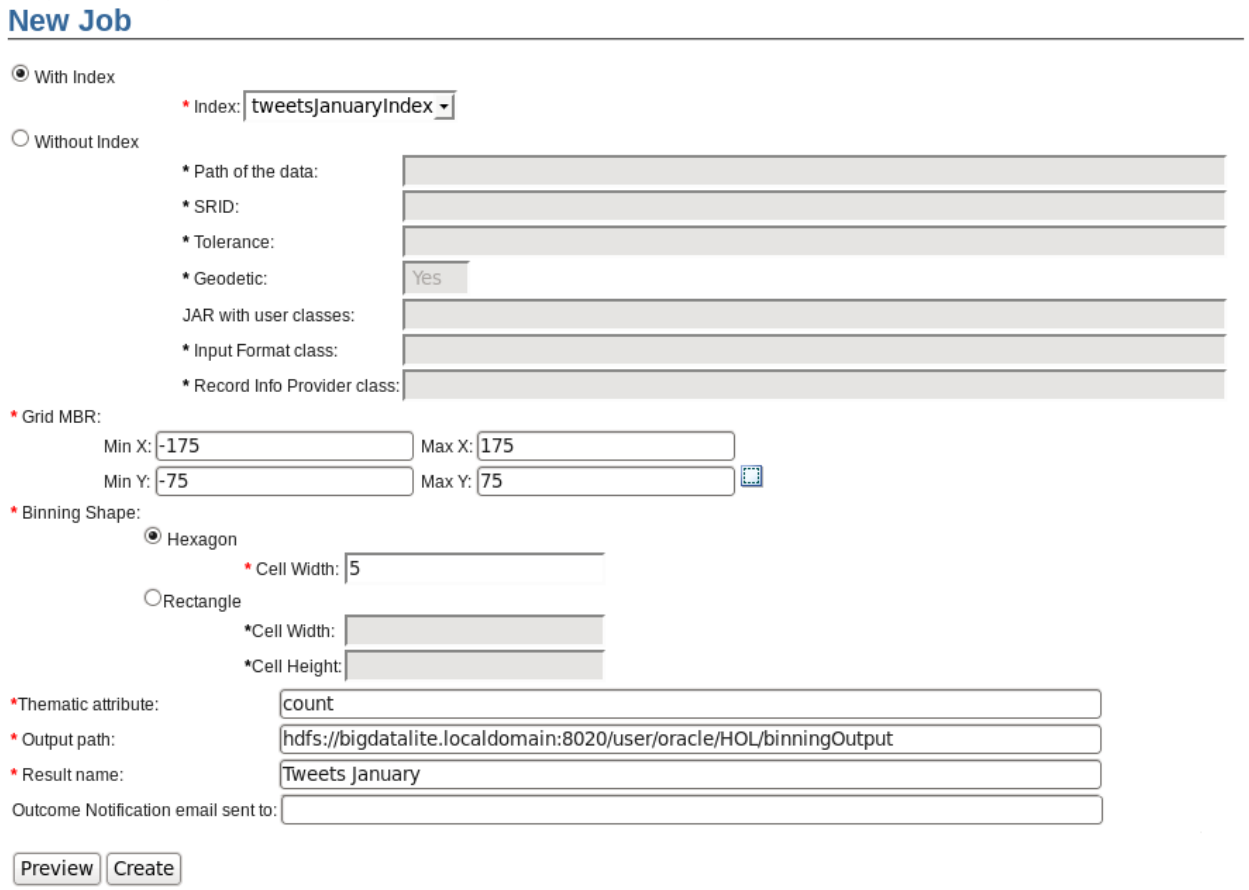
5. Click the button  to see the analysis area.



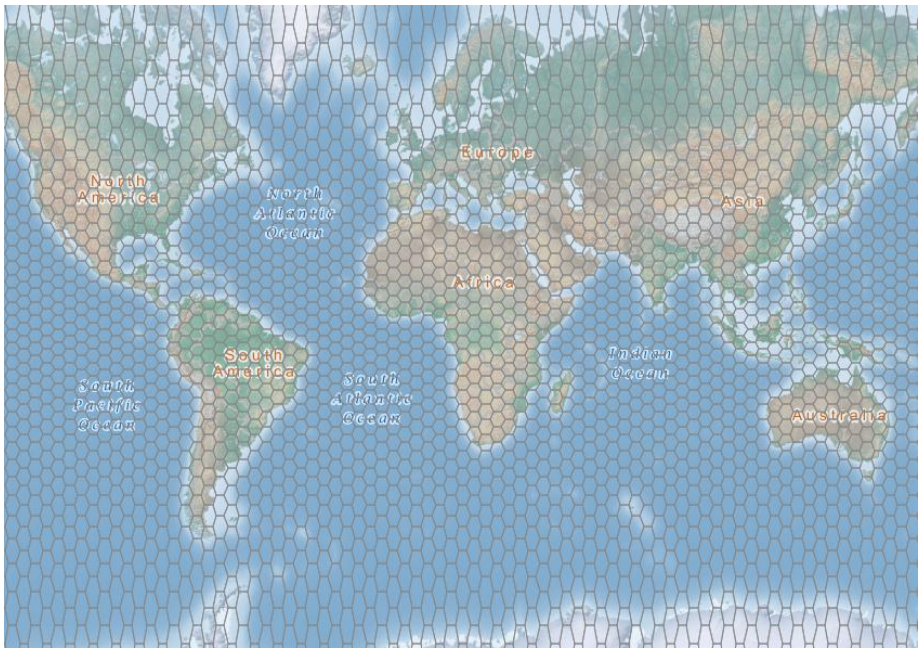


6. The binning shape will be hexagon and the width will be 5.
7. Let the thematic map as count. The count attribute means that each cell in the grid will contain as information the number of tweets that are in it.

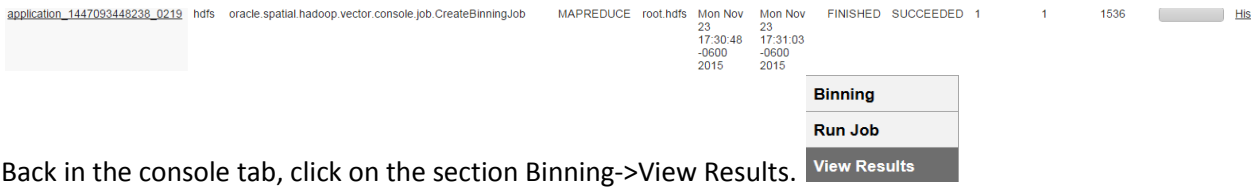
8. Specify the required fields:
    a. Output path: The Hadoop job output path. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/binningOutput*
    b. Result Name: Tweets January



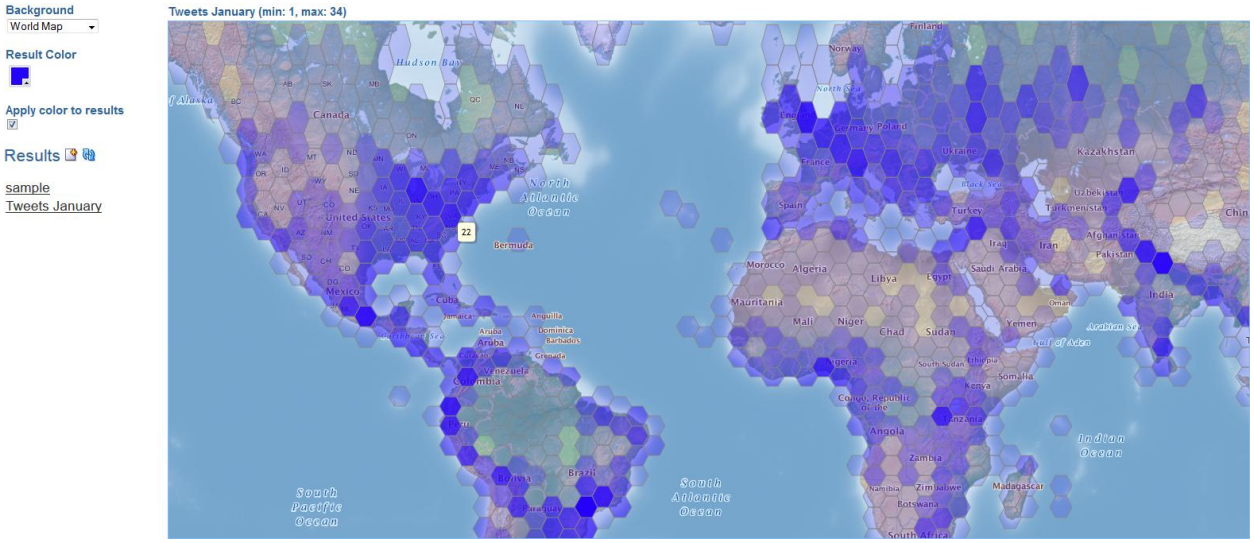
9. Click Preview and see the preview of the grid.



10. Click Create.
11. Open http://localhost:8088/cluster/apps in a new Firefox tab and wait until the job is completed successfully.



12. Back in the console tab, click on the section Binning->View Results. Note that the result Tweets January can take an extra minute to be created. Refresh the screen with F5 after a minute.
13. Click on the result Tweets January.

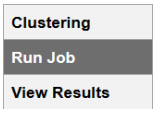


14. See the number of tweets in each cell moving the mouse cursor on it.

Run a Clustering Job

In this task we will run a clustering analysis on the tweets. The K –means clustering method is used and the number of clusters is 2. The K-means method is popular for clustering analysis in data mining. More information about it can be found here https://en.wikipedia.org/wiki/K-means_clustering.

1. Open http://localhost:8045/spatialviewer/
2. Click on the section Clustering->Run Job.



3. Specify all the required details:
    a. Path of data: Provide the HDFS data path. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/tweets.json*
    b. The SRID of the geometries: 8307
    c. The tolerance of the geometries: The tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors). For this example we set 0.5
    d. If the geometries are geodetic or not: Yes

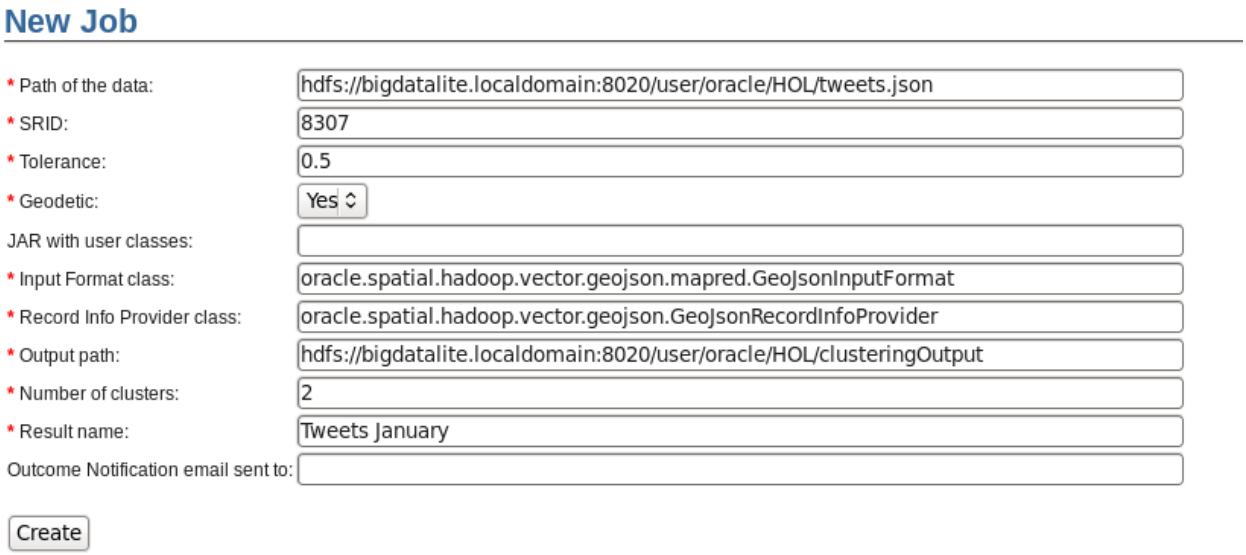
e. Input Format class: The InputFormat class implementation used to read the input data. For this example we set oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat.

f. Record Info Provider class: The class that will provide the spatial information. For this example we set *oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider.*

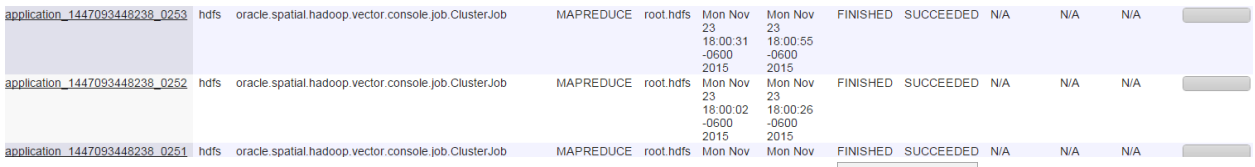
g. Output path: The Hadoop job output path. For this example we set *hdfs://bigdatalite.localdomain:8020/user/oracle/HOL/clusteringOutput*

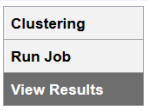
h. Number of clusters: 2

i. Result Name: Tweets January

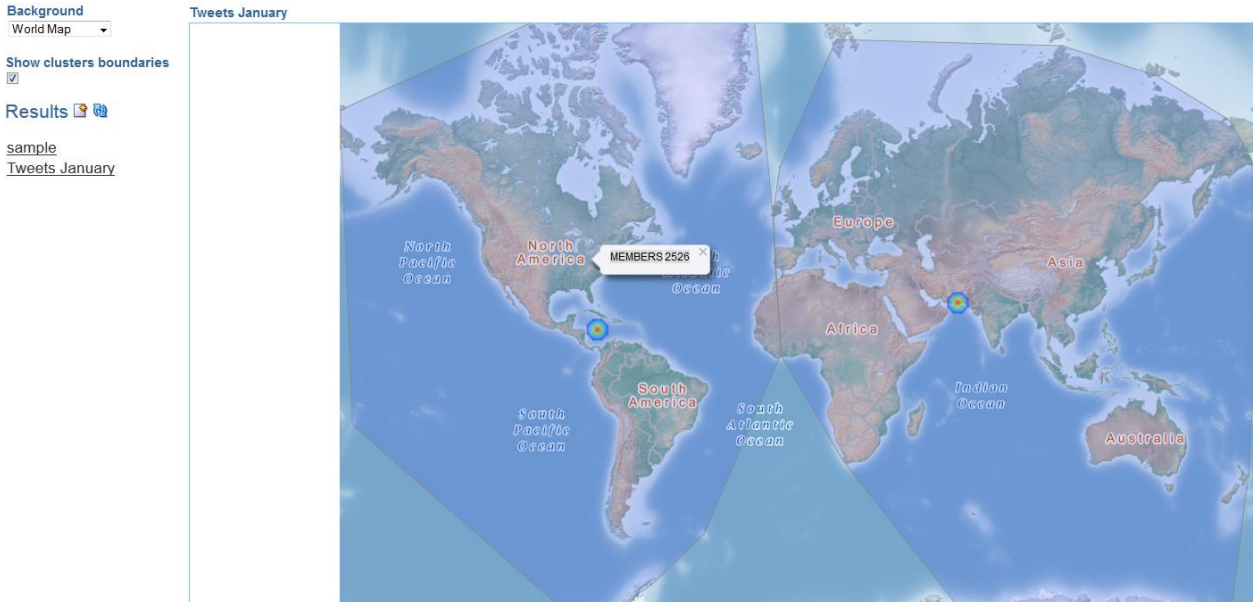


4. Click Create.
5. Open http://localhost:8088/cluster/apps in a new Firefox tab and wait until the jobs are completed successfully. Note that this analysis uses the K-Means algorithm and will run several jobs (about 9 jobs).





6. Back in the console tab, click on the section Clustering->View Results.
7. Click on the result *Tweets January.*
8. Select the checkbox Show clusters boundaries.



9. See the clusters centers  and boundaries. Clicking on a cluster will show the number of tweets inside that cluster.

Lab Part 2: Use Oracle Big Data Spatial and Graph Vector Command Line

In the prior section we showed how to run jobs and display their results in the vector console. Now let´s see how to run jobs from the command line. For simplicity the categorization job and binning job will run without spatial index.

In this section we will use the command line to:

1) Create Spatial Index with GeoJSON file (2 mins)
2) Run a Categorization Job using a custom layer (8 mins)
3) Run a Binning Job using a custom RecordInfoProvider (7 mins)

You can find the additional classes used in those examples in the folder `/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/java/src/`

More examples can be found in the folder `/opt/oracle/oracle-spatial-graph/spatial/vector/examples`.

Step 1: Open the terminal. (3 mins for the following steps)

Step 2: We will take advantage of the `libjars` option in the `hadoop jar` command to make the API needed JAR's available to the map and reduce tasks. For that end, create an environment variable named `HADOOP_LIB_JARS` that reference those jars by typing the following commands in the terminal:

```
export API_LIB_DIR=/opt/oracle/oracle-spatial-graph/spatial/vector/jlib

export HADOOP_LIB_JARS=$API_LIB_DIR/sdohadoop-
vector.jar,$API_LIB_DIR/sdoapi.jar,$API_LIB_DIR/sdoutl.jar,$API_LIB_DIR/ojdbc
8.jar
```

Step 3: Make these same JAR's available to the client JVM, which is the JVM that's created when you run the `hadoop jar` command. For this to happen, you should set the `HADOOP_CLASSPATH` environment variable containing the needed jars:

```
export HADOOP_CLASSPATH=$API_LIB_DIR/sdohadoop-
vector.jar:$API_LIB_DIR/sdoapi.jar:$API_LIB_DIR/sdoutl.jar:$API_LIB_DIR/ojdbc
8.jar:$HADOOP_CLASSPATH
```

Note: Since our examples don't use the MVSuggest data enrichment service we don't need to include all the jars in `/opt/oracle/oracle-spatial-graph/spatial/vector/jlib`.

Note: The API provides InputFormats and RecordInfoProvider implementation for the common formats GeoJSON and ESRI Shapefiles. It is possible to use any Hadoop provided or customized InputFormat and any customized RecordInfoProvider.

Create Spatial Index with GeoJSON file

This task creates a spatial index using the job `oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing` with the file `/user/oracle/HOL/tweets.json` as input. The name of the new index is `indexGeoJSON` and the metadata of the index will be located in the HDFS directory `/user/oracle/HOL/indexMetadataDir`. The needed arguments of the job are:

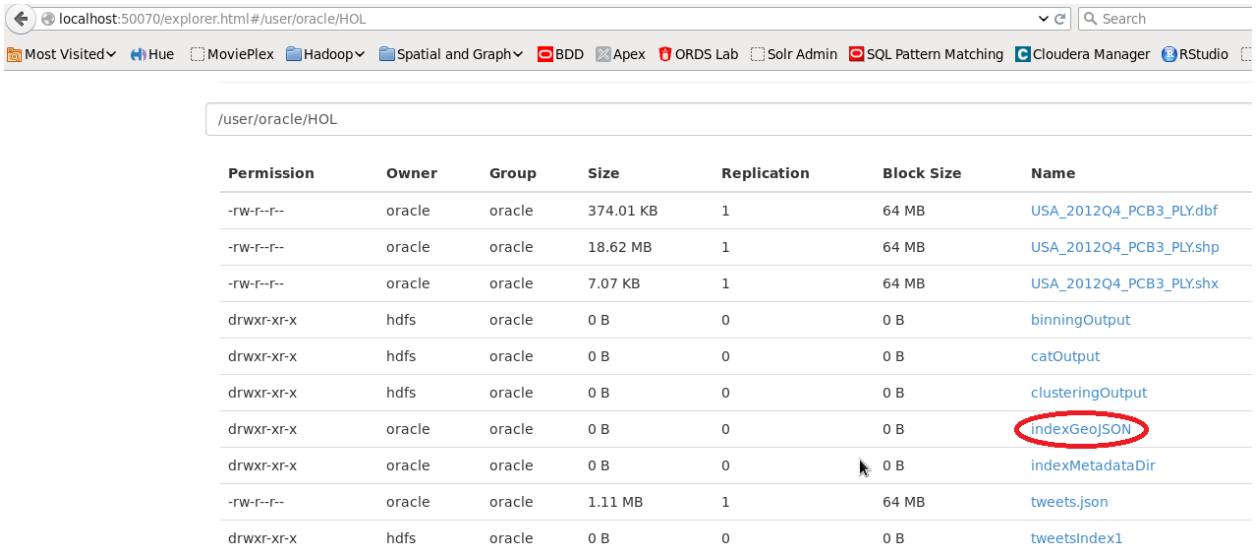
- input : the location of the data to be indexed.
- output: the location of the resulting spatial index.
- inputFormat: the InputFormat class implementation used to read the input data.
- recordInfoProvider: the RecordInfoProvider implementation used to extract information from the records read by the InputFormat class.
- srid: the Spatial Reference System id of the spatial data
- geodetic: value that indicates whether the geometries are geodetic or not
- tolerance: double value which represents the tolerance used when performing spatial operations (tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors))

- indexName: the name of the index to be generated
- indexMetadataDir: the directory where the spatial index metadata will be stored in HDFS.
- overwriteIndexMetadata: argument that indicates whether the index metadata can be overwritten if an index with the same name already exists.

1. Type the command in the terminal:

```
hadoop jar $API_LIB_DIR/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.SpatialIndexing -libjars
$HADOOP_LIB_JARS input=/user/oracle/HOL/tweets.json
output=/user/oracle/HOL/indexGeoJSON
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFor
mat
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordIn
foProvider srid=8307 geodetic=true tolerance=0.5 indexName=indexGeoJSON
indexMetadataDir=/user/oracle/HOL/indexMetadataDir
overwriteIndexMetadata=true
```

2. Once the job is finished, confirm that the index was created by opening up the Firefox browser and going to the index HDFS location using this link: http://localhost:50070/explorer.html#/user/oracle/HOL.



Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	oracle	oracle	374.01 KB	1	64 MB	USA_2012Q4_PCB3_PLY.dbf
-rw-r--r--	oracle	oracle	18.62 MB	1	64 MB	USA_2012Q4_PCB3_PLY.shp
-rw-r--r--	oracle	oracle	7.07 KB	1	64 MB	USA_2012Q4_PCB3_PLY.shx
drwxr-xr-x	hdfs	oracle	0 B	0	0 B	binningOutput
drwxr-xr-x	hdfs	oracle	0 B	0	0 B	catOutput
drwxr-xr-x	hdfs	oracle	0 B	0	0 B	clusteringOutput
drwxr-xr-x	oracle	oracle	0 B	0	0 B	indexGeoJSON
drwxr-xr-x	oracle	oracle	0 B	0	0 B	indexMetadataDir
-rw-r--r--	oracle	oracle	1.11 MB	1	64 MB	tweets.json
drwxr-xr-x	hdfs	oracle	0 B	0	0 B	tweetsIndex1

Run a Categorization Job using a custom layer

This task will categorize the tweets by countries and regions inside the eurozone. The API doesn't provide the data sets of the countries in the eurozone so we have to provide them. The job used is `oracle.spatial.hadoop.vector.mapred.job.Categorization`. The needed arguments of the job are:

- spatialOperation: the spatial operation to perform between the input data set and the hierarchical data set. Allowed values are IsInside and AnyInteract
- input : the location of the input data in HDFS.
- output: the path where the results will be stored in HDFS
- inputFormat: the InputFormat class implementation used to read the input data.
- recordInfoProvider: the RecordInfoProvider implementation used to extract information from the records read by the InputFormat class.
- srid: the Spatial Reference System id of the spatial data
- geodetic: value that indicates whether the geometries are geodetic or not
- tolerance: double value which represents the tolerance used when performing spatial operations (tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors))
- hierarchyInfo: the fully qualified name of a HierarchyInfo implementation. It defines the structure of the current hierarchy data. An implementation example is shown further in this example.
- hierarchyIndex: the HDFS path where the hierarchy data index will be placed. This index is used by the job to avoid finding parent-children relationships each time is required.
- hierarchyDataPaths: a comma separated list of paths of the hierarchy data. The paths should be sorted in ascending way by hierarchy level.

1. Before executing the job we have to create the `HierarchyInfo` class and make it available for our job. In this example the class is already created and is contained in the jar `/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/jlib/eurohierarchyinfo.jar`.

   The class fully qualified name is `hol.EuroHierarchyInfo` and the implementation is:

```
/**
 * Subclass of DynaAdminHierarchyInfo that setups the hierarchy composed of the layers
 * eurozone_countries and eurozone_provinces.
 *
 */
public class EuroHierarchyInfo extends DynaAdminHierarchyInfo {

    public EuroHierarchyInfo(){
        super();
        addLevel(
                1, //hierarchy level number
                "eurozone_countries", //hierarchy level name
                "_id", //path to the JSON field used to associate entries of this level with their children.
                null
                );

        addLevel(
                2, //hierarchy level number
                "eurozone_provinces", //hierarchy level name
                null,
                "properties.ISO" //path to the JSON field used to associate entries of this level with their parent.
                );
    }
}
```

The class `hol.EuroHierarchyInfo` describes the hierarchy that will be used to categorize the tweets data. The first hierarchy is the eurozone_countries and the second hierarchy is the eurozone_provinces. Both files are in the folder `/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/data`. In the file eurozone_countries.json each country has a field _id that refers to the property ISO in the provinces records of the file eurozone_provinces.json.

Example of record in the file eurozone_countries.json:


```
{"type":"Feature","_id":"FRA","geometry":{"type":"Polygon","coord
inates":[[1.44136,42.60366,1.47851,42.65168,…,1.44136,42.60366]]}
,"properties":{"Continent":"EU","Name":"France","Alt_Region":"EME
A","Country Code":"FRA"},"label_box":[-
1.12061,45.13915,6.02255,49.19591]}
```


And example of a record representing a France province in eurozone_provinces.json:


```
{"type":"Feature","_id":"3023519","geometry":{"type":"Polygon","c
oordinates":[[9.19977,41.36465,9.25876,…,41.36465]]},"properties"
:{"Country":"France","ISO":"FRA","State Province
Name":"Corse","Country Code_State Province
Name":"FRA_Corse","Country Name_State Province
Name":"France_Corse"},"label_box":[8.70974,42.08453,9.53075,42.54
732]}
```


By setting this information class the API will know that any record of a France province is also a record belonging to the France country without extra calculation. If those data are not provided then the provinces that are spatially inside a country are considered a province of it.

2. The next step is to make the jar `eurohierarchyinfo.jar` available for Hadoop. To do that we will add it to the environment variables `HADOOP_CLASSPATH` and `HADOOP_LIB_JARS` typing the following commands in the terminal:

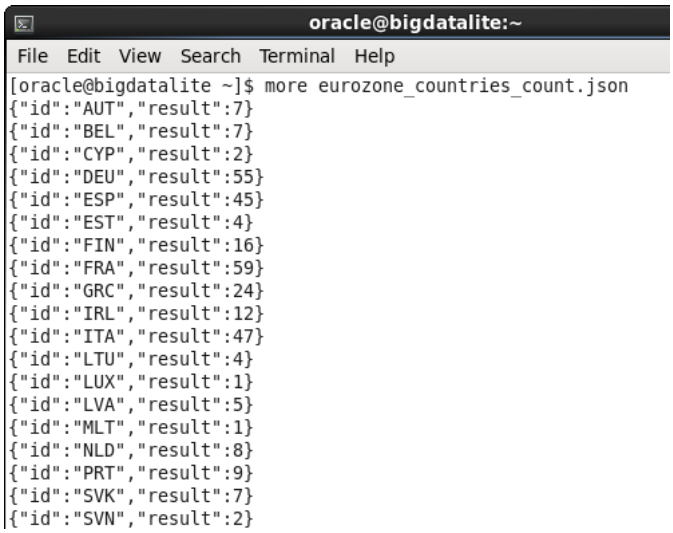
```
export HADOOP_CLASSPATH=/opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/jlib/eurohierarchyinfo.jar:$HADOOP_CLASS
PATH

export HADOOP_LIB_JARS=/opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/jlib/eurohierarchyinfo.jar,$HADOOP_LIB_J
ARS
```

3. Now run the categorization job entering the following command in the terminal:

```
hadoop jar $API_LIB_DIR/sdohadoop-vector.jar
oracle.spatial.hadoop.vector.mapred.job.Categorization -libjars
$HADOOP_LIB_JARS spatialOperation=IsInside
input=/user/oracle/HOL/tweets.json output=/user/oracle/HOL/catOutput
inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFor
mat
```

```
recordInfoProvider=oracle.spatial.hadoop.vector.geojson.GeoJsonRecordIn
foProvider srid=8307 geodetic=true tolerance=0.5
hierarchyInfo=hol.EuroHierarchyInfo
hierarchyIndex=/user/oracle/HOL/hierarchyIndex
hierarchyDataPaths=file:///opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/data/eurozone_countries.json,file:///opt/oracl
e/oracle-spatial-graph/spatial/vector/HOL/data/eurozone_provinces.json
```

4. Once the job completes, the result has been saved in the HDFS folder `/user/oracle/HOL/catOutput`. Let´s copy it locally. Type the command `hadoop fs -get /user/oracle/HOL/catOutput/*.json .`
5. Type:
   `ls`
   Two files have been copied `eurozone_countries_count.json` and `eurozone_provinces_count.json`.
6. Review `eurozone_countries_count.json` with the command:
   `more eurozone_countries_count.json`



Each record id refers to the record in the eurozone_countries.json file, for example the result:

```
{"id":"FRA","result":59}
```

Refers to the record in the file eurozone_countries.json:

```
{"type":"Feature","_id":"FRA","geometry":{"type":"Polygon","coord
inates":[[1.44136,42.60366,1.47851,42.65168,…,1.44136,42.60366]]}
,"properties":{"Continent":"EU","Name":"France","Alt_Region":"EME
A","Country Code":"FRA"},"label_box":[-
1.12061,45.13915,6.02255,49.19591]}
```

Meaning that 59 tweets have been sent from France.

Run a Binning Job using a custom RecordInfoProvider

This task will run a binning analysis on the tweets. That is a query area will be split into cells that are the bins, then each bin (cell) will contain the information of the average number of followers of the tweets in the bin. To do this we will create a custom `RecordInfoProvider` that will return as extra field the number of followers that is needed in this analysis. The job used is `oracle.spatial.hadoop.vector.mapred.job.Binning`. The required arguments for this job are:

- input : the location of the input data in HDFS.
- output: the path where the results will be stored in HDFS
- inputFormat: the InputFormat class implementation used to read the input data.
- recordInfoProvider: the RecordInfoProvider implementation used to extract information from the records read by the InputFormat class.
- srid: the Spatial Reference System id of the spatial data
- geodetic: value that indicates whether the geometries are geodetic or not
- tolerance: double value which represents the tolerance used when performing spatial operations (tolerance reflects the distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors))
- cellShape: the shape of the cells. It can be RECTANGLE or HEXAGON
- cellSize: the size of the cells in the format width,height
- gridMbr: the minimum and maximum dimension values for the grid in the form minX,minY,maxX,maxY
- aggrFields: a comma-separated list of field names that will be aggregated to the result.

7. Before executing the job we have to create the `RecordInfoProvider` class and make it available for our job. In this example the class is already created and is contained in the jar `/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/jlib/tweetsrecordinfoprovider.jar`.

   The class fully qualified name is `hol.TweetsRecordInfoProvider` and the implementation is:

```
/**
 * RecordInfoProvider implementation that parse a GeoJSON text. Additionally to provide geometry it adds
 * the followers_count property to the extra fields list.
 *
 */
public class TweetsRecordInfoProvider implements RecordInfoProvider<LongWritable, Text>{
    private ObjectMapper jsonMapper = null;
    private JsonNode recordNode = null;
    /**
     * Creates a new instance
     */
    public TweetsRecordInfoProvider(){
        //create a unique json mapper to parse all the records
        jsonMapper = new ObjectMapper();
    }

    @Override
    public void setCurrentRecord(LongWritable key, Text value) {
        try{
            //parse the current value
            recordNode = jsonMapper.readTree(value.toString());
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    @Override
    public String getId() {
        //returns the id of the record
        return recordNode.get("_id").getTextValue();
    }

    @Override
    public JGeometry getGeometry() {
        //returns the geometry of the record
        return JsonUtils.readGeometry(
                        recordNode.get("geometry"), //the GeoJSON geometry
                        2, //the dimensions of the geometry
                        8307 //the SRID of the geometry
                        );
    }
    @Override
    public boolean getExtraFields(Map<String, String> extraFields) {
        //Find the followers count
        String followersCount = recordNode.get("properties").get("followers_count").getValueAsText();
        //and add it to the extra fields that will be available when performing an analysis
        extraFields.put("followers_count", followersCount);

        //return true since we added an extra field
        return true;
    }
}
```

The class `hol.TweetsRecordInfoProvider` provides the geometry of each record and adds to the extra fields list the followers_count that will be use in the analysis.

Example of record in the file tweets.json:

{"type":"Feature","_id":"1","geometry":{"type":"Point","coordinates":[121.31111,24.98889]},"properties":{"followers_count":82,"friends_count":120,"location":"Taiwan"}}

8. The next step is to make the jar `tweetsrecordinfoprovider.jar` available for Hadoop. To do that we will add it to the environment variables `HADOOP_CLASSPATH` and `HADOOP_LIB_JARS` typing the following commands in the terminal:

```
export HADOOP_CLASSPATH=/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/jlib/tweetsrecordinfoprovider.jar:$HADOOP_CLASSPATH

export HADOOP_LIB_JARS=/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/jlib/tweetsrecordinfoprovider.jar,$HADOOP_LIB_JARS
```

9. Now run the binning job entering the following command in the terminal:

```
hadoop jar $API_LIB_DIR/sdohadoop-vector.jar oracle.spatial.hadoop.vector.mapred.job.Binning -libjars $HADOOP_LIB_JARS input=/user/oracle/HOL/tweets.json output=/user/oracle/HOL/binningOutput inputFormat=oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat recordInfoProvider=hol.TweetsRecordInfoProvider srid=8307 geodetic=true tolerance=0.5 cellShape=HEXAGON cellSize=5 gridMbr=-175,-85,175,85 aggrFields=followers_count
```

10. The result has been saved in the HDFS folder `/user/oracle/HOL/binningOutput`. Let´s copy it locally. Type the command

```
hadoop fs -get /user/oracle/HOL/binningOutput/bin_res .
```

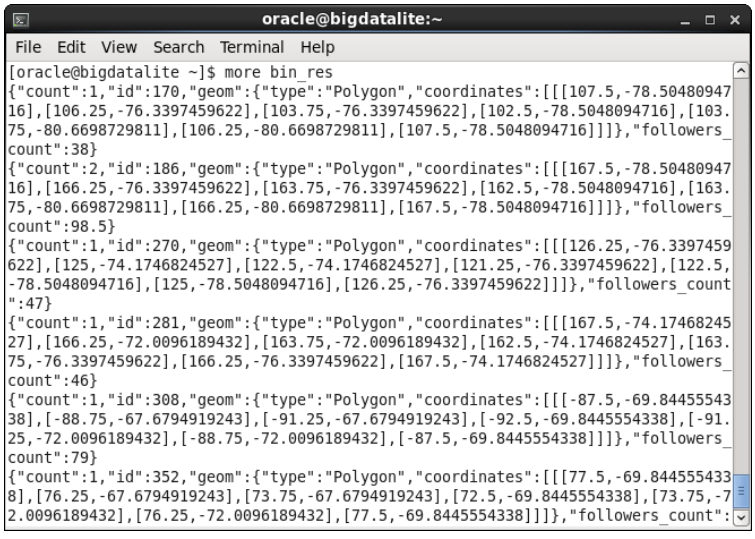
11. Type:

```
ls
```

The file `bin_res` has been copied.

12. Review `bin_res` with the command:

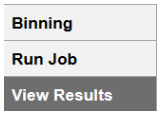
```
more bin_res
```

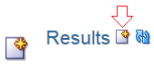


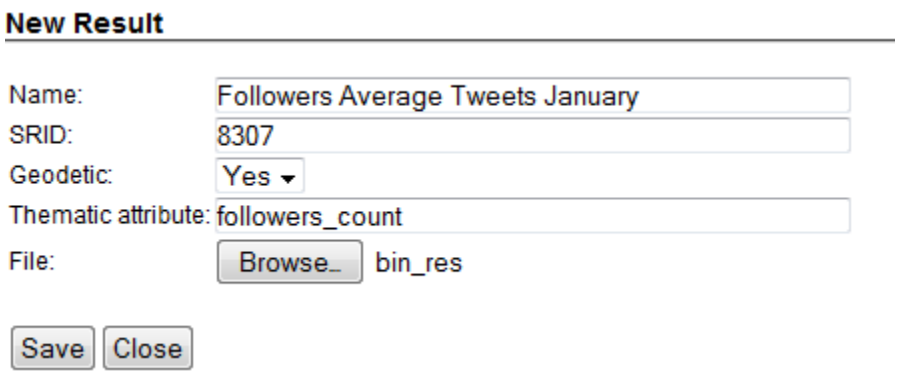
Each record contains an `id`, the number of tweets in the bin (`count`), the geometry (`geom`) of the bin that is a HEXAGON and the average number of followers for the tweets inside the bin (`follower_count`), for example the result:

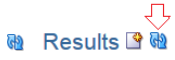

```
{"count":1,"id":444,"geom":{"type":"Polygon","coordinates":[[[66.25,-67.6794919243],[65,-65.5144284149],[62.5,-65.5144284149],[61.25,-67.6794919243],[62.5,-69.8445554338],[65,-69.8445554338],[66.25,-67.6794919243]]]},"followers_count":98}
}
```

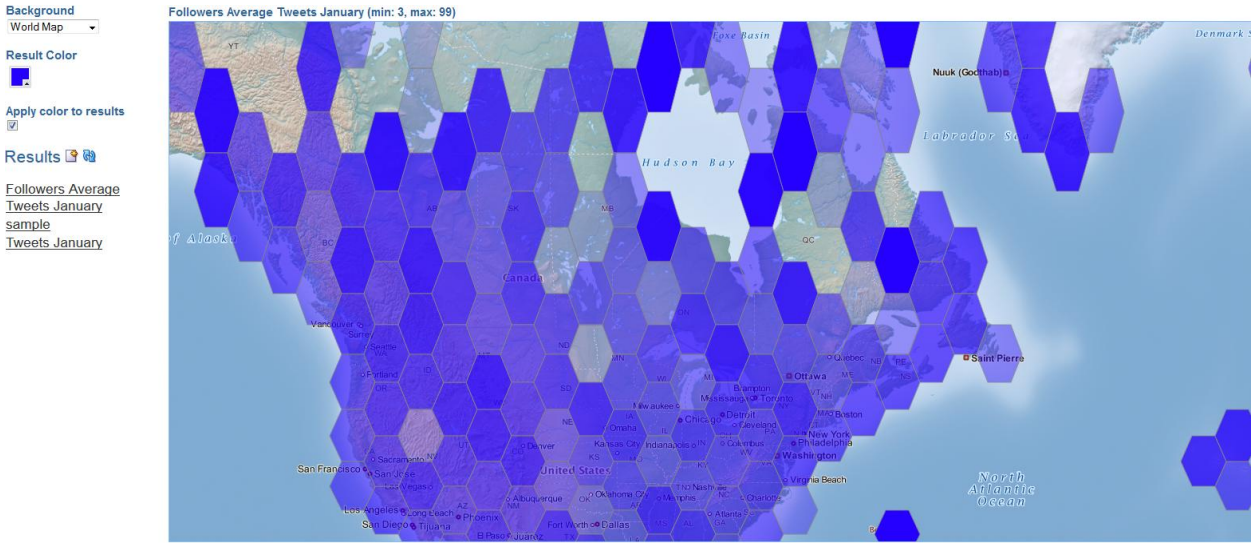

13. To see the result on the console open http://localhost:8045/spatialviewer/
14. Click on the section Binning->View Results.



15. Click on the add result button  .
16. Specify all the required details:
    a. Name: Followers Average Tweets January
    b. SRID: 8307
    c. Geodetic: Yes
    d. Thematic attribute: followers_count
    e. Select the file bin_res



17. Click Save
18. When the message “Result has been added successfully” appears click Close.
19. Click on the refresh button  .
20. Click on the result Followers Average Tweets January.



Lab Part 3: Create Customized Jobs Using the Oracle Big Data Spatial and Graph Vector API

So far we saw how to use the console features and some API included jobs. Now we will see how to create our own jobs in java using the API framework.

In this section we will:

1) Create a Job to Filter Spatial and Non Spatial Data (11 mins)
2) Create a Job to Calculate Polygons length using an ESRI Shapefile (8 mins)

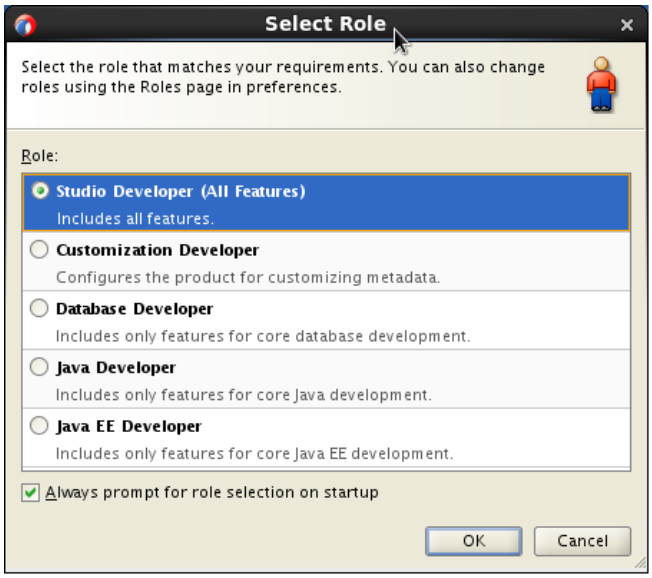
For simplicity in the examples we won't use any spatial index.

To learn how to filter with index and more about how to create jobs using the java API review the examples in the folder `/opt/oracle/oracle-spatial-graph/spatial/vector/examples`.

You can find the additional classes used in those examples in the folder `/opt/oracle/oracle-spatial-graph/spatial/vector/HOL/java/src/`

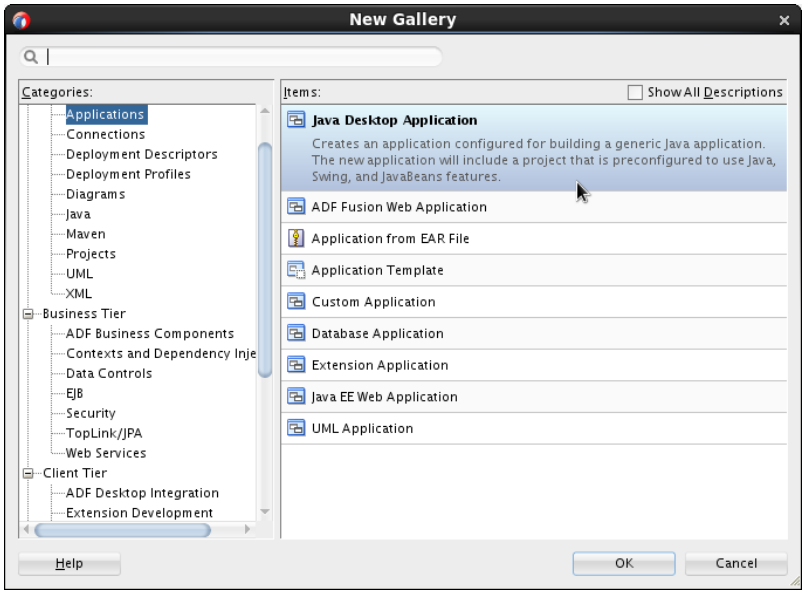
Step 1: Open JDeveloper by clicking the JDev icon in the upper Task Bar (7 minutes for the following steps).

Step 2: Select the role Studio Developer (All Features) and click OK.

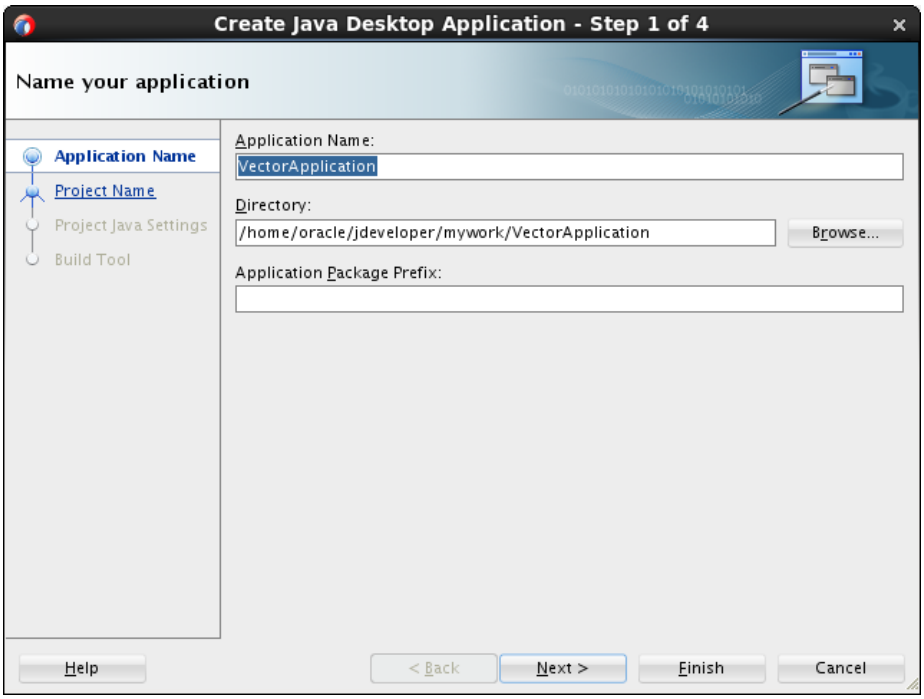


Step 2: Go to the Application menu and from the submenu list select New....

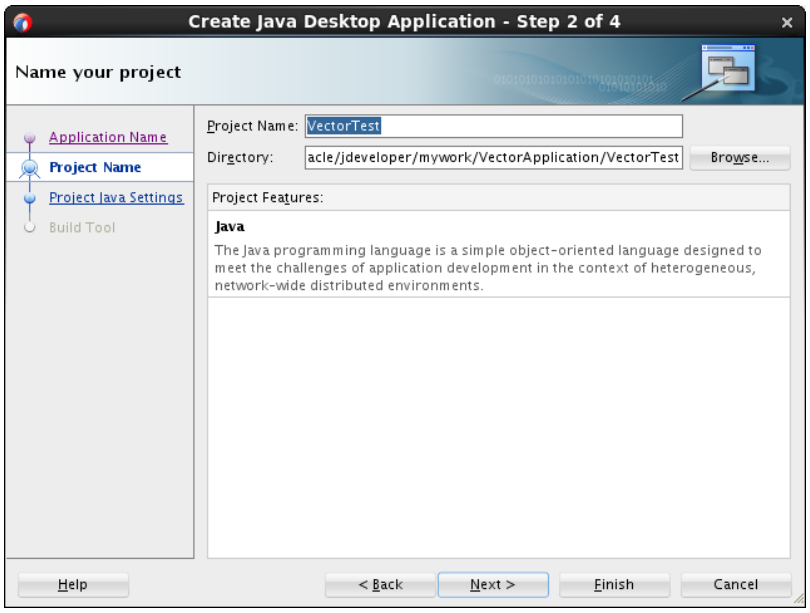
Step 3: Select Java Desktop Application and click OK



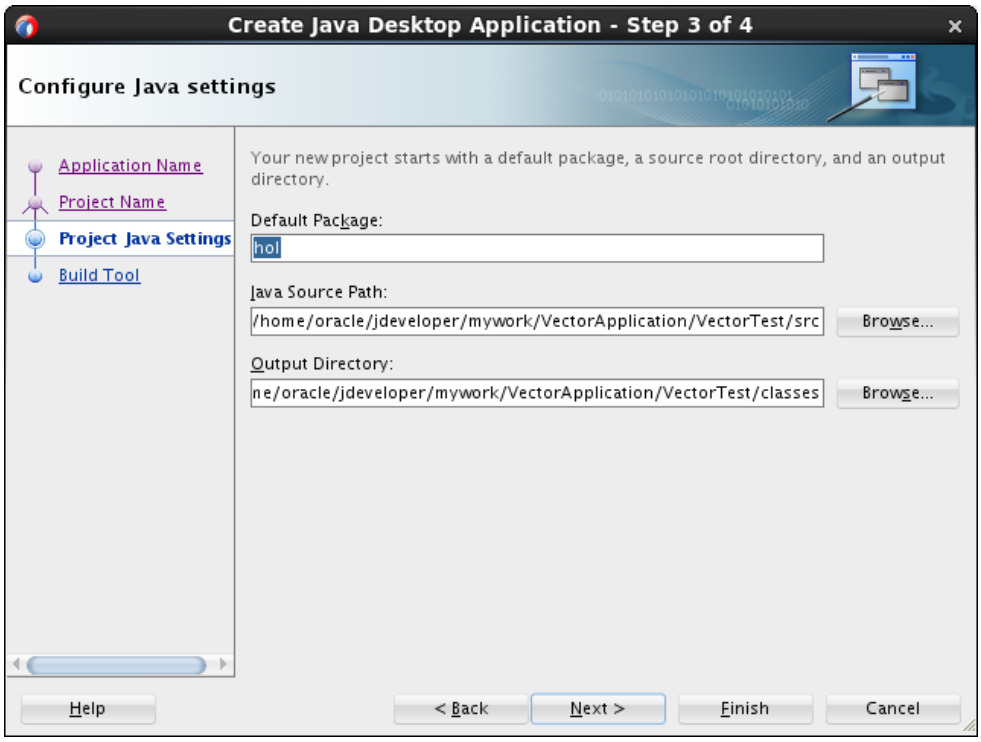
Step 4: Set the Application Name to *VectorApplication* and click Next.



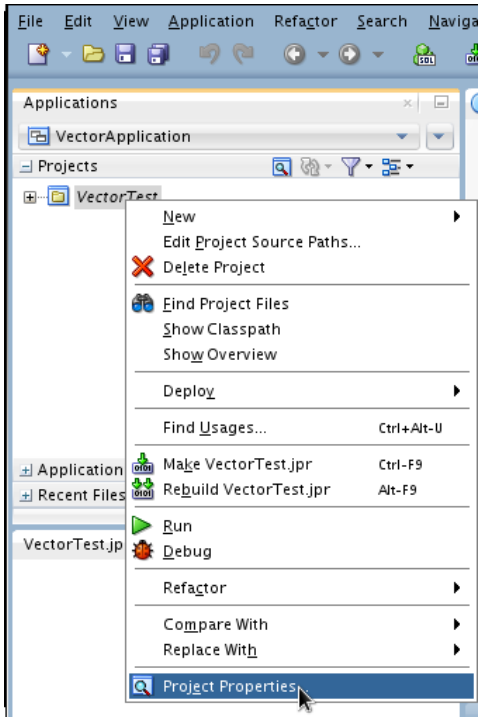
Step 5: Set the project name to VectorTest and click Next.



Step 4: Set the default package to hol and click Finish.



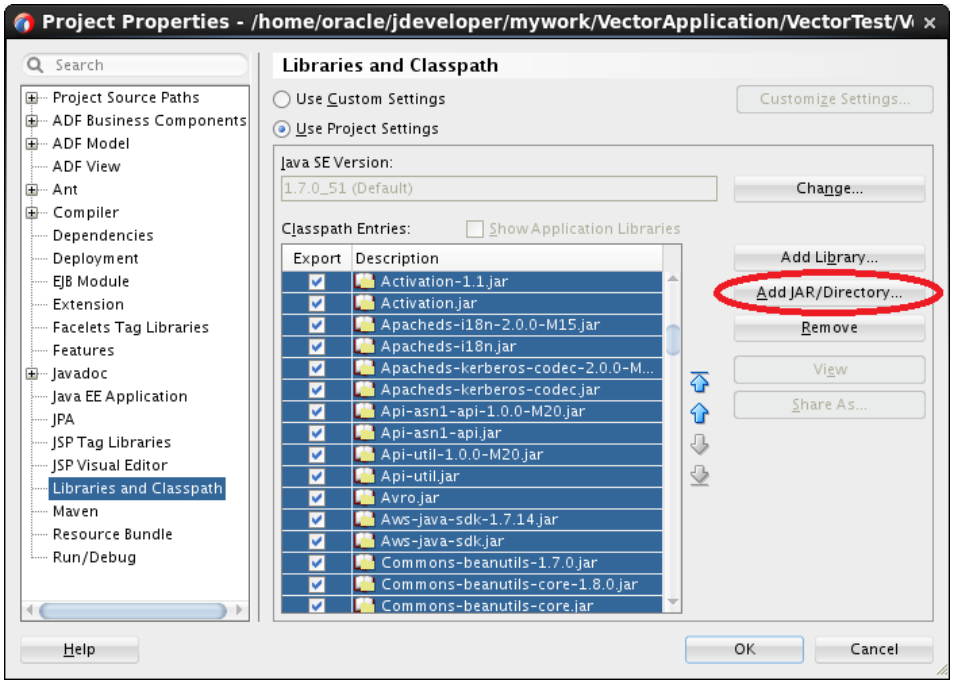
Step 5: Right click on the VectorTest project in the Application Navigator and select Project Properties.



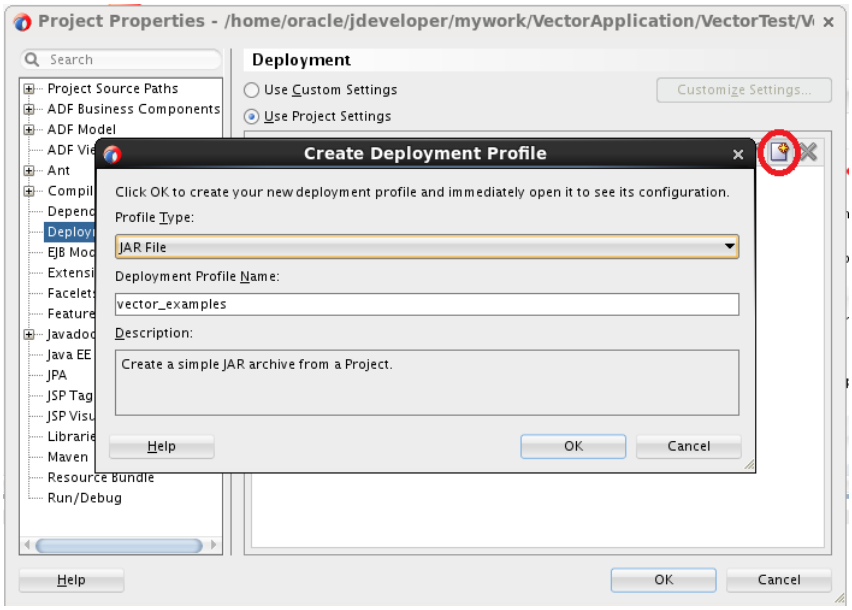
Select the Libraries and Classpath option and click on Add JAR/Directory.

In the Add Archive/Directory window set the path to `/opt/oracle/oracle-spatial-graph/spatial/vector/jlib` and select all the jar files in this folder. Click Open. Then in the same

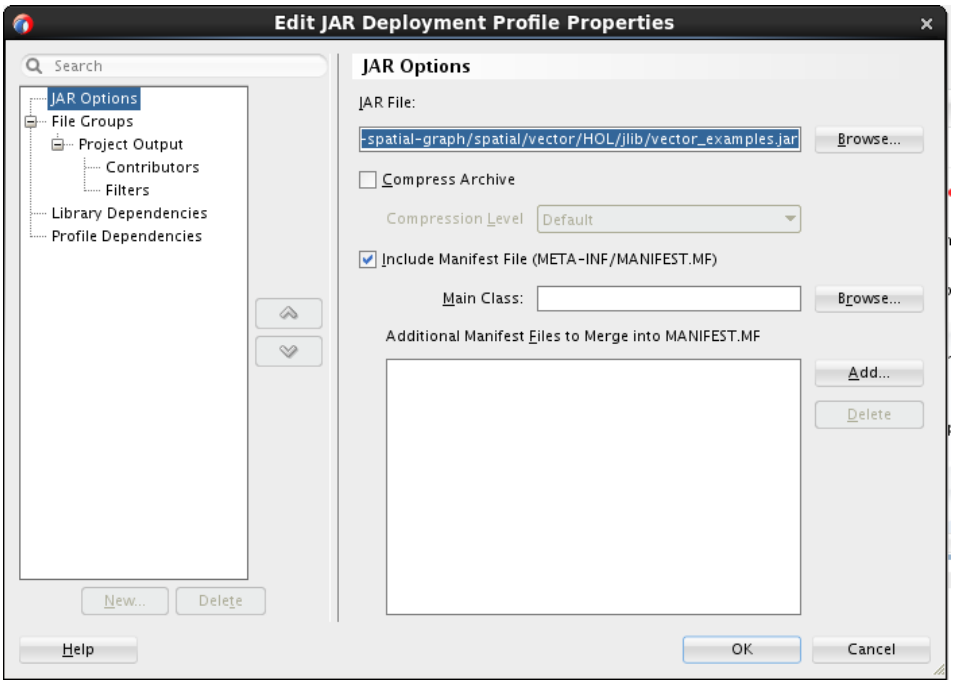
way add all the Hadoop jars located in `/usr/lib/hadoop/client`. Now the project knows about the required files for vector processing. Finally Click OK.



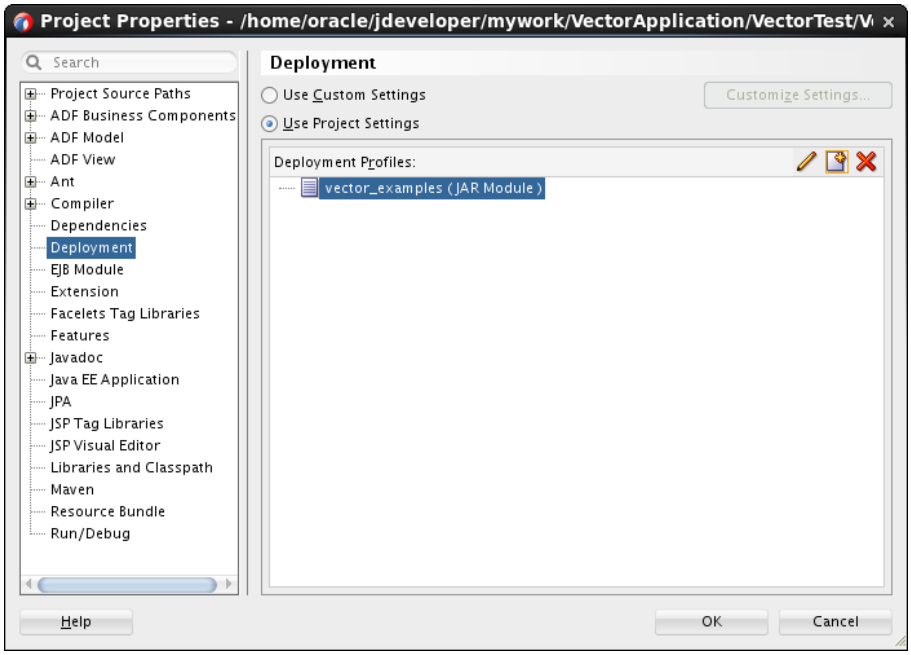
Step 6: Right click on the project and select Project Properties. In the emergent window select Deployment option and click on Create Deployment Profile icon. Select JAR File as the Profile Type and set vector_examples as the Deployment Profile Name, click OK to Finish.



Step 7: In the emergent Edit JAR Deployment Profile Properties set the jar file to /opt/oracle/oracle-spatial-graph/spatial/vector/HOL/jlib/vector_examples.jar and click OK.



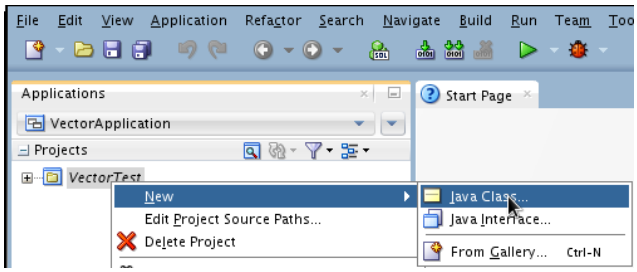
Step 8: Finally click OK in the Deployment screen.



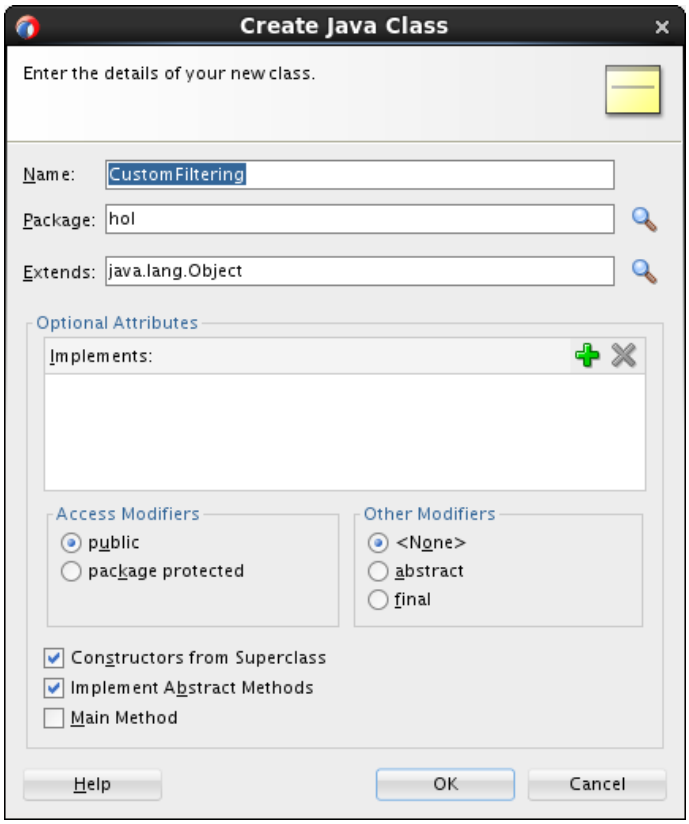
Create a Job to Filter Spatial and Non Spatial Data

The job will filter the tweets that are inside a query window and that have a number of followers higher than 50.

1. Right click on the VectorTest project in the Application Navigator and select New. In the New Gallery window select Java class.



In the Create Java Class window, set *CustomFiltering* as the Name for the class and click Ok.



2. First add the imported classes to avoid compilation errors:

```
import java.io.IOException;

import oracle.spatial.hadoop.vector.geojson.GeoJsonRecordInfoProvider;
import oracle.spatial.hadoop.vector.geojson.mapred.GeoJsonInputFormat;
```

```
import oracle.spatial.hadoop.vector.mapred.input.SpatialFilterInputFormat;
import oracle.spatial.hadoop.vector.util.JobUtils;
import oracle.spatial.hadoop.vector.util.SpatialOperation;
import oracle.spatial.hadoop.vector.util.SpatialOperationConfig;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.ObjectMapper;
```

3. Let´s start programming the CustomFiltering.java class creating the main function (The Map class is provided in step 4).

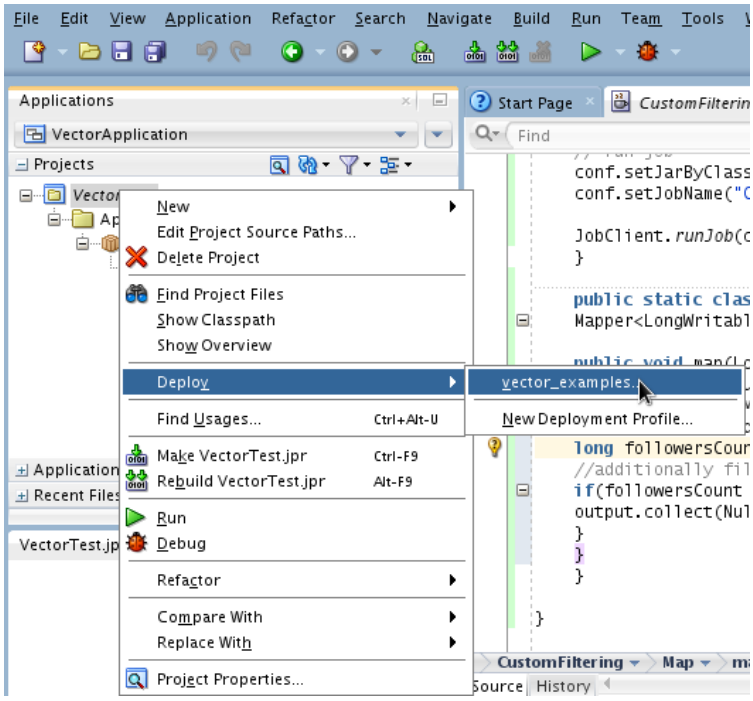
```
public static void main(String[] init_args) throws Exception{
    Configuration config = new Configuration();

    // This step is important as init_args contains ALL the arguments passed to hadoop on the
    // command line (such as -libjars [jar files]).  What's left after .getRemainingArgs is just the
    // application specific arguments
    String [] args = new GenericOptionsParser(config, init_args).getRemainingArgs();

    JobConf conf = new JobConf(config);

    //set input path and format
    JobUtils.setupInputPath(args[0], conf);
    //InputFormat that will filter spatially the data
    conf.setInputFormat(SpatialFilterInputFormat.class);

    //set internal input format
    SpatialFilterInputFormat.setInternalInputFormatClass(conf, GeoJsonInputFormat.class);
    //as no spatial index is used a RecordInfoProvider is needed
    SpatialFilterInputFormat.setRecordInfoProviderClass(conf, GeoJsonRecordInfoProvider.class);

    //set spatial operation used to filter the records
    SpatialOperationConfig spatialOperationConfig = new SpatialOperationConfig();
    spatialOperationConfig.setSrid(8307);
    spatialOperationConfig.setTolerance(0.5);
    spatialOperationConfig.setGeodetic(true);
    spatialOperationConfig.setOperation(SpatialOperation.IsInside);
    //set the query window (the query window covers an area in the North of Mexico)
    spatialOperationConfig.setJsonQueryWindow("{\"type\":\"Polygon\", \"coordinates\":[[-106, 25, -106, 30, -104, 30, -104, 25, -106, 25]]}");

    spatialOperationConfig.store(conf);

    // output path
    JobUtils.setupOutputPath(args[1], null, conf);

    // output format
    conf.setOutputFormat(TextOutputFormat.class);
    // mapper
```

```
        conf.setMapperClass(Map.class);
        conf.setMapOutputKeyClass(NullWritable.class);
        conf.setMapOutputValueClass(Text.class);
        //one reducer will be used to avoid multiple outputs
        conf.setNumReduceTasks(1);

        // run job
        conf.setJarByClass(CustomFiltering.class);
        conf.setJobName("CustomFiltering example");

        JobClient.runJob(conf);
}
```

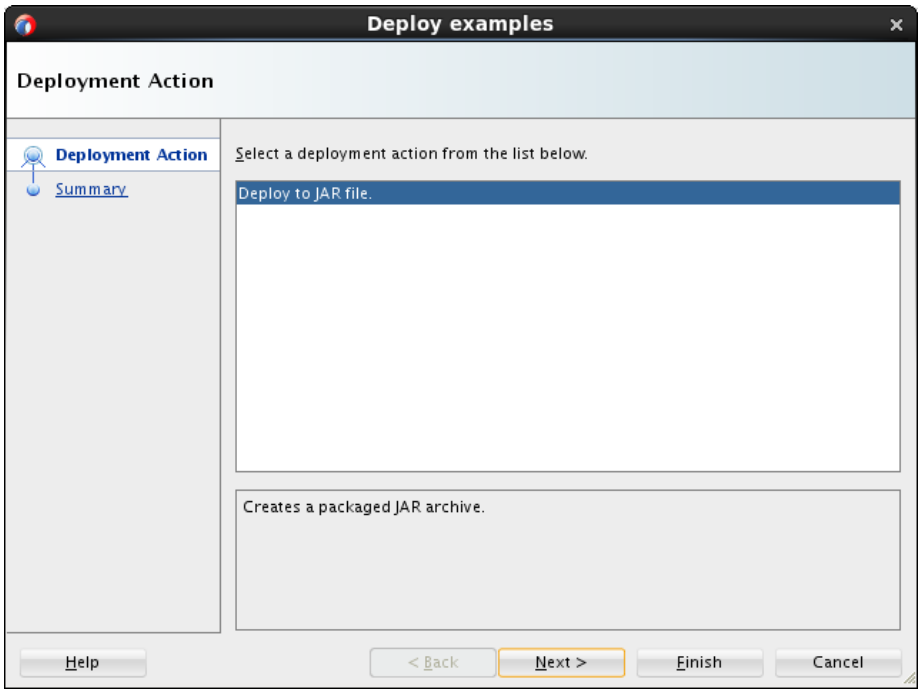
4. Add the Map class to the CustomFiltering class. The mapper will filter by tweet followers.

```
public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, NullWritable, Text> {

                public void map(LongWritable key, Text value,
                        OutputCollector<NullWritable, Text> output, Reporter reporter) throws IOException {
                        ObjectMapper jsonMapper = new ObjectMapper();
                        JsonNode recordNode = jsonMapper.readTree(value.toString());
                        long followersCount = recordNode.get("properties").get("followers_count").getLongValue();
                        //additionally filter the tweets with more than 50 followers
                        if(followersCount > 50){
                                output.collect(NullWritable.get(), value);
                        }
                }
}
```

5. The code is ready!
6. Right click on the project select Deploy and the vector_examples profile you just created.



7. Click Finish. The Jar file is deployed and ready to use.



8. Open the terminal.

9. We will take advantage of the `libjars` option in the `hadoop jar` command to make the API needed JAR's available to the map and reduce tasks. For that end, create an environment variable named `HADOOP_LIB_JARS` that references those jars by typing the following command in the terminal:

```
export API_LIB_DIR=/opt/oracle/oracle-spatial-graph/spatial/vector/jlib

export HADOOP_LIB_JARS=$API_LIB_DIR/sdohadoop-
vector.jar,$API_LIB_DIR/sdoapi.jar,$API_LIB_DIR/sdoutl.jar,$API_LIB_DIR
/ojdbc8.jar
```

10. Make these same JAR's available to the client JVM, which is the JVM that's created when you run the `hadoop jar` command. For this to happen, you should set the `HADOOP_CLASSPATH` environment variable containing the needed jars:

```
export HADOOP_CLASSPATH=$API_LIB_DIR/sdohadoop-
vector.jar:$API_LIB_DIR/sdoapi.jar:$API_LIB_DIR/sdoutl.jar:$API_LIB_DIR
/ojdbc8.jar:$HADOOP_CLASSPATH
```

11. Now run the job by entering the following command in the terminal:

```
hadoop jar /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/jlib/vector_examples.jar hol.CustomFiltering -
```

```
libjars $HADOOP_LIB_JARS /user/oracle/HOL/tweets.json
/user/oracle/HOL/filteringOutput
```

12. The result has been saved in the HDFS folder `/user/oracle/HOL/filteringOutput`. Let´s copy it locally. Type the commands

```
mkdir filteringOutput
cd filteringOutput
hadoop fs -get /user/oracle/HOL/filteringOutput/* .
```

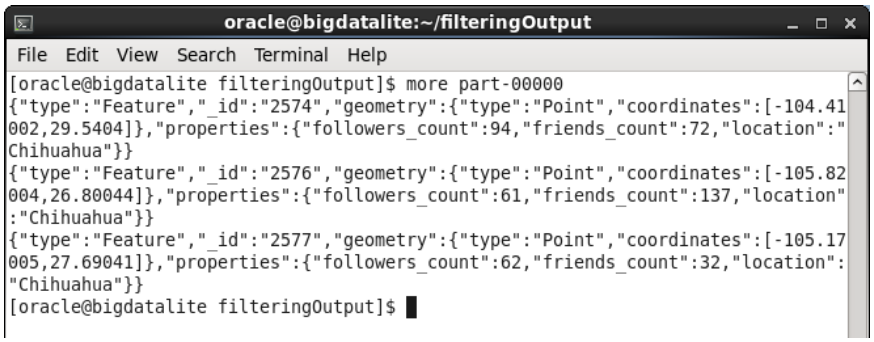
13. Type:

```
ls
```

The file `part-00000` has been copied.

14. Review part-00000 with the command:

```
more part-00000
```

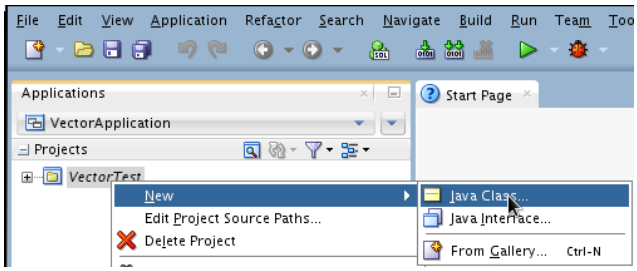


Three results are displayed. The resulting records have the same id and properties than in the input file tweets.json and were selected because they have more than 50 followers as specified in our custom filter class.

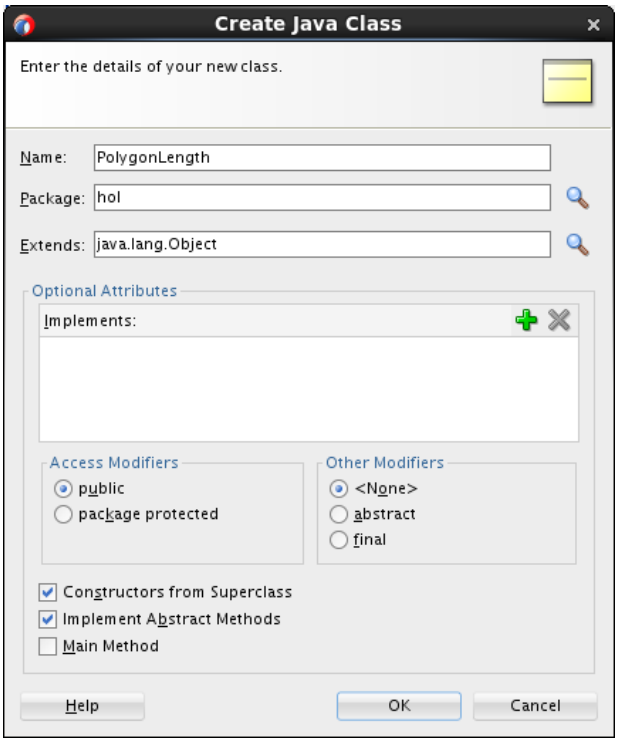
Create a Job to Calculate Polygons length using an ESRI Shapefile

The job receives as input an ESRI Shapefile containing polygons. The polygons represent the 3-digit postal boundaries of the USA. The output will be the record attributes and the length in meters of the polygons.

1. Right click on the VectorTest project in the Application Navigator and select New. In the New Gallery window select Java class.



In the Create Java Class window, set *PolygonLength* as the Name for the class and click Ok.



2. First add the imported classes to avoid compilation errors:

```
import java.io.IOException;
import java.text.DecimalFormat;

import oracle.spatial.geometry.J3D_Geometry;
import oracle.spatial.geometry.JGeometry;
import oracle.spatial.hadoop.vector.RecordInfo;
import oracle.spatial.hadoop.vector.mapred.input.CompositeInputFormat;
import oracle.spatial.hadoop.vector.mapred.input.FileSplitInputFormat;
import oracle.spatial.hadoop.vector.mapred.input.RecordInfoLoader;
import oracle.spatial.hadoop.vector.shapefile.ShapeFileRecordInfoProvider;
import oracle.spatial.hadoop.vector.shapefile.mapred.ShapeFileInputFormat;
import oracle.spatial.hadoop.vector.util.ConfigParams;
import oracle.spatial.hadoop.vector.util.JobUtils;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.NullWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
```

3. Let´s start programming the PolygonLength.java class creating the main function (The Map class is provided in step 4).

```
public static void main(String[] init_args) throws Exception{
    Configuration config = new Configuration();
    // This step is important as init_args contains ALL the arguments passed to hadoop on the command
    // line (such as -libjars [jar files]).  What's left after .getRemainingArgs is just the
    // application specific arguments
    String [] args = new GenericOptionsParser(config, init_args).getRemainingArgs();

    JobConf conf = new JobConf(config);
    conf.setJarByClass(PolygonLength.class);
    conf.setJobName("PolygonLength example");
    //one reducer will be used
    conf.setNumReduceTasks(1);

    //Setting add extra fields to true means that all the fields available in the Shapefile will
    //be available when the records will be process in the mapper.
    ShapeFileRecordInfoProvider.setAddExtraFields(conf, true);

    //set input and output paths in HDFS
    JobUtils.setupInputPath(args[0], conf);
    JobUtils.setupOutputPath(args[1], null, conf);

    //set the mapper class
    conf.setMapperClass(Map.class);

    conf.setMapOutputKeyClass(NullWritable.class);
    conf.setMapOutputValueClass(Text.class);
    conf.setOutputKeyClass(NullWritable.class);
    conf.setOutputValueClass(RecordInfo.class);
    //A CompositeInputFormat that only provides the source FileSplits as its values so mapreduce
    //components can read the data directly using its internal input format.
    conf.setInputFormat(FileSplitInputFormat.class);

    //set SRID of the geometries
    conf.setInt(ConfigParams.SRID, 4326);

    //set the input format class
    CompositeInputFormat.setInternalInputFormatClass(conf, ShapeFileInputFormat.class);
    //set the record info provider class
    CompositeInputFormat.setRecordInfoProviderClass(conf, ShapeFileRecordInfoProvider.class);
    //set the output format class
    conf.setOutputFormat(TextOutputFormat.class);
    //run the job
    JobClient.runJob(conf);
}
```

4. Add the Map class to the PolygonLength class.

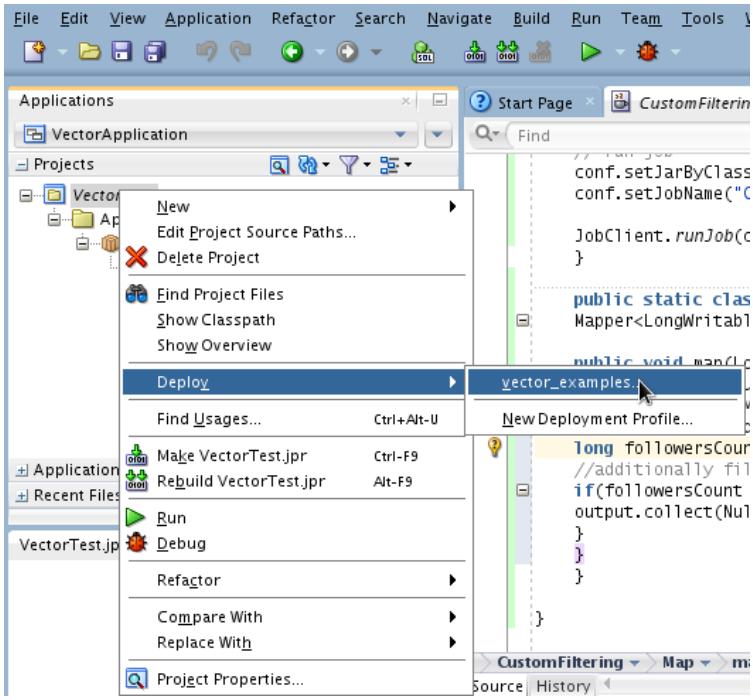
   Notes:

   - The mappers will process up to 10 records to reduce the time of the job.
   - Inverse flattening and semi-Major Axis is information specific to the coordinate system. For example with the SRID 4326 that is used in the example the information is available here: https://en.wikipedia.org/wiki/World_Geodetic_System

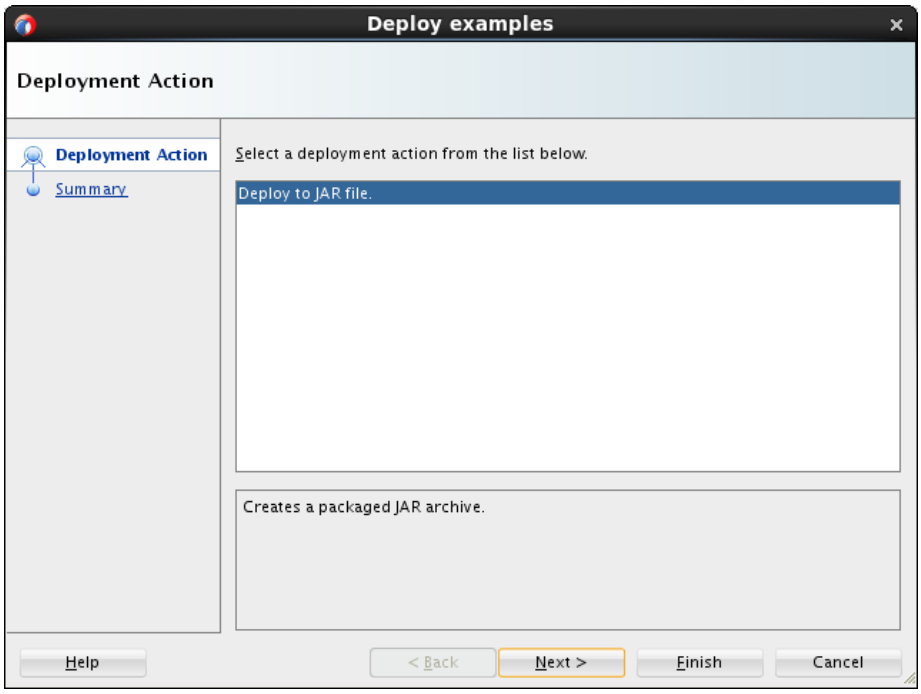
```
public static class Map extends MapReduceBase implements Mapper<NullWritable, FileSplit, NullWritable, Text>{
        private RecordInfoLoader<Writable, Writable> riLoader = null;

        @Override
        public void configure(JobConf conf) {
                super.configure(conf);
                //create the record info loader
                riLoader = new RecordInfoLoader<Writable, Writable>(conf);
        }

        @Override
        public void map(NullWritable key, FileSplit value,
                OutputCollector<NullWritable, Text> output, Reporter reporter)
                                                        throws IOException {
                //set a maximum number of results to process to reduce the time of the job
                int maxNumberOfResults = 10;
                int currentResultNumber = 1;
                //Semi-Major Axis (Equatorial Radius)
                double WGS84_SMAX = 6378137;
                //inverse flattening
                double WGS84_IFLAT = 298.257223563;

                riLoader.startLoading(value, reporter);

                while(riLoader.hasNext()){
                        if(currentResultNumber > maxNumberOfResults){
                                //return since the maximum number of results to process has been reached
                                return;
                        }

                        currentResultNumber++;

                        //get the RecordInfo
                        RecordInfo ri = riLoader.next();

                        if(ri!=null && ri.getGeometry() != null){
                                try {
                                        //the class J3D_Geometry contains the length function. It works with
                                        //both 3D and 2D geometries.
                                        J3D_Geometry geometry = JGeometry.make_3dgeom(ri.getGeometry(),
                                                false, //doesn't ignore SRID conversion
                                                ri.getGeometry().getSRID(), //geometries SRID
                                                0 //height that is 0 since in our case the geometry is 2D
                                        );

                                        double length = geometry.length(
                                                0.05 //tolerance
                                                , "TRUE" //is_g3d TRUE means that the data are geodetic
```

```
                                        , WGS84_SMAX
                                        , 1/WGS84_IFLAT
                                        , 1 //use 1 for the unit-of-measure factor (to return meters)
                                );

                                //collect the information to print as result
                                StringBuffer sb = new StringBuffer();

                                sb.append("-----------------------------------");
                                sb.append("\npostal code reference : " +ri.getField("postcode"));
                                sb.append("\nCountry abbreviation : " +ri.getField("iso_ctry"));
                                sb.append("\nCountry : " +ri.getField("admin1"));
                                sb.append("\nState : " +ri.getField("admin2"));
                                sb.append("\nCounty : " +ri.getField("admin3"));
                                sb.append("\nCity : " +ri.getField("admin4"));
                                sb.append("\nState abbreviation : " +ri.getField("state"));

                                sb.append("\nlength : " +new DecimalFormat(".##").format(length));
                                sb.append("\n-----------------------------------");
                                //collect the result
                                output.collect(NullWritable.get(), new Text(sb.toString()));

                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

15. The code is ready!
16. Right click on the project select Deploy and the vector_examples profile you just created.



5. Click Finish. The Jar file is deployed and ready to use.



6. Open the terminal.

7. We will take advantage of the `libjars` option in the `hadoop jar` command to make the API needed JAR's available to the map and reduce tasks. For that end, create an environment variable named `HADOOP_LIB_JARS` that references those jars typing the following command in the terminal:

```
export API_LIB_DIR=/opt/oracle/oracle-spatial-graph/spatial/vector/jlib

export HADOOP_LIB_JARS=$API_LIB_DIR/sdohadoop-
vector.jar,$API_LIB_DIR/sdoapi.jar,$API_LIB_DIR/sdoutl.jar,$API_LIB_DIR
/ojdbc8.jar
```

8. Make these same JAR's available to the client JVM, which is the JVM that's created when you run the `hadoop jar` command. For this to happen, you should set the `HADOOP_CLASSPATH` environment variable containing the needed jars:

```
export HADOOP_CLASSPATH=$API_LIB_DIR/sdohadoop-
vector.jar:$API_LIB_DIR/sdoapi.jar:$API_LIB_DIR/sdoutl.jar:$API_LIB_DIR
/ojdbc8.jar:$HADOOP_CLASSPATH
```

9. Now run the job by entering the following command in the terminal:

```
hadoop jar /opt/oracle/oracle-spatial-
graph/spatial/vector/HOL/jlib/vector_examples.jar hol.PolygonLength -
```

```
libjars $HADOOP_LIB_JARS /user/oracle/HOL/USA_2012Q4_PCB3_PLY.shp
/user/oracle/HOL/lengthResult
```

10. The result has been saved in the HDFS folder `/user/oracle/HOL/lengthResult`. Let´s copy it locally. Type the commands

```
mkdir lengthResult
cd lengthResult
hadoop fs -get /user/oracle/HOL/lengthResult/* .
```

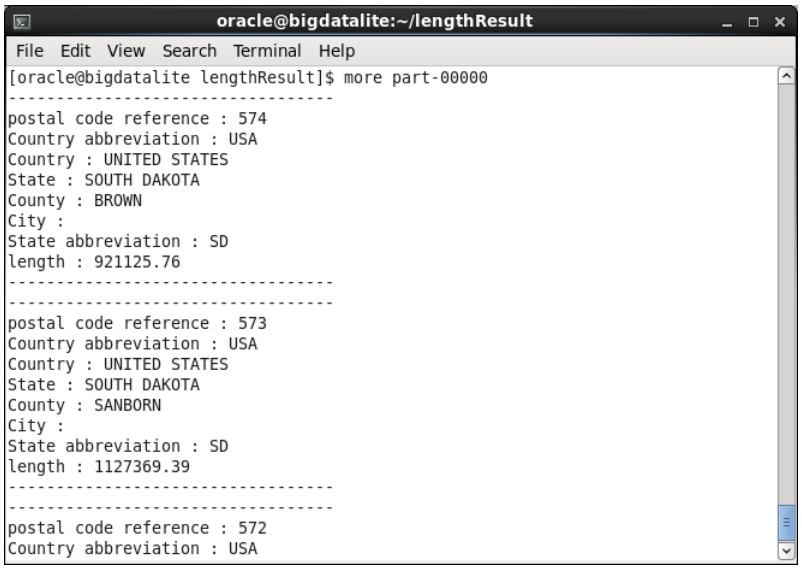
11. Type:

```
ls
```

The file `part-00000` has been copied.

12. Review part-00000 with the command:

```
more part-00000
```



The resulting records have the record attributes and the length of the polygons in meters.

Lab Part 4: Use Oracle Big Data Spatial and Raster Console

The following lessons will walk us through various steps that are needed to load images into HDFS, then select the images for processing and create mosaic images and finally add custom functionality to the processing using the Raster Console - ImageServer.

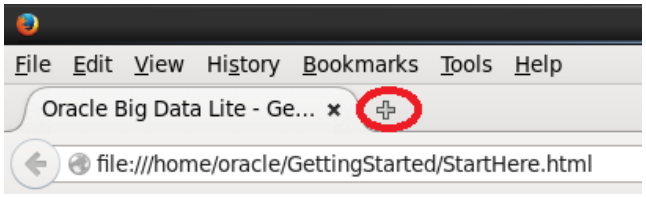
Lab Exercises:

1) Load raster images to HDFS and create basic mosaic (10 mins)

Let´s get started:

Step 1: Open the web browser Firefox by clicking the icon below.

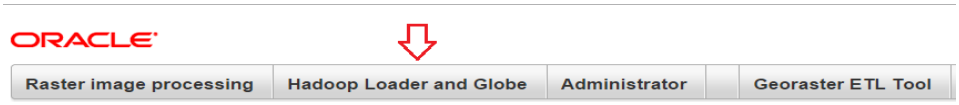
Note that if you need to open a new tab in Firefox click 



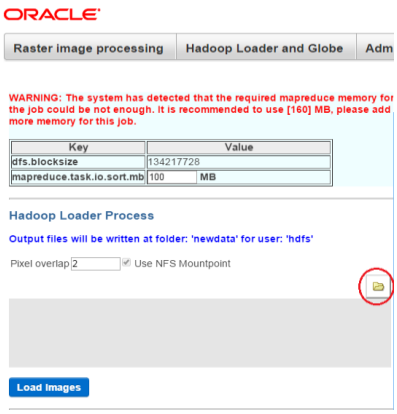
Load raster images to HDFS and create basic mosaic

In this section we will learn how to locate images from the file system, and select those to be loaded into hdfs, a whole folder can also be selected and all images inside it will be loaded as well.

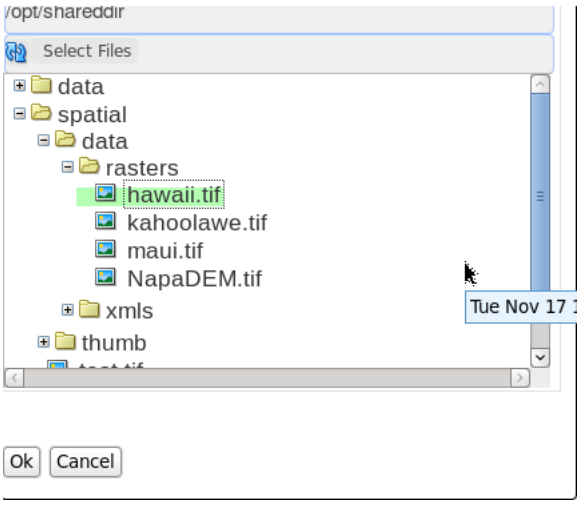
1. Open http://localhost:8045/imageserver/#
2. Go to Hadoop Loader and Globe tab.



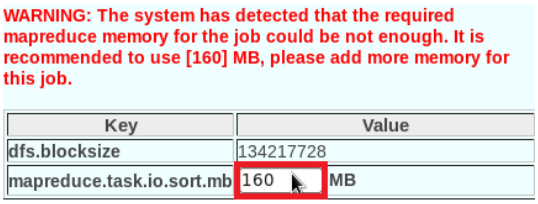
3. Configure the task memory as recommended by the message.



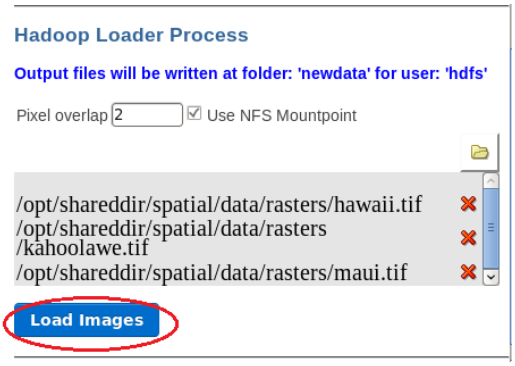
4. Go the left panel and click on the **open** folder button and select all the Hawaii images one at a time, **Hawaii.tif, Kahoolawe.tif and maui.tif**:



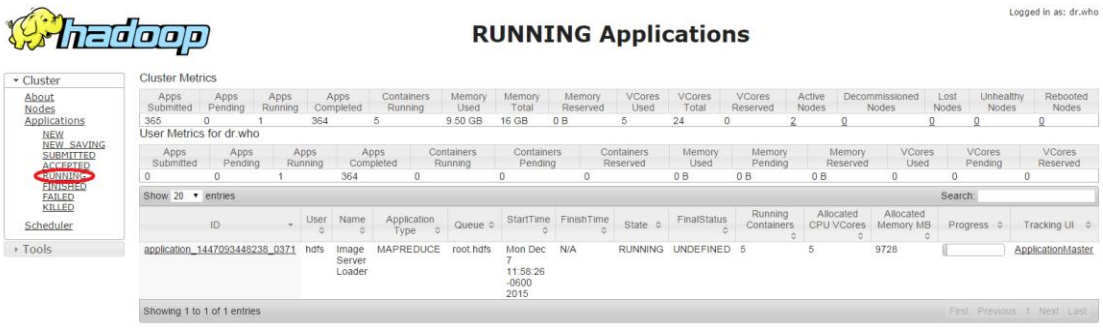
5. Do not forget to configure memory as necessary.



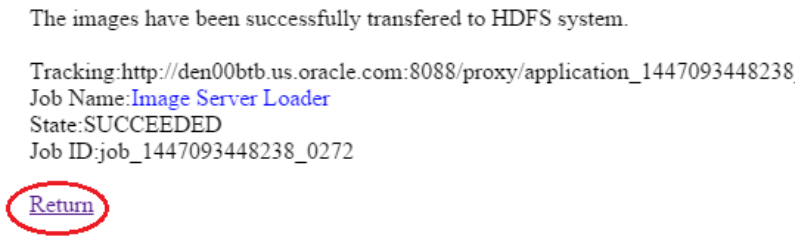
6. Click on **Load Images** button.



7. Open a new tab in your web browser and type http://localhost:8088/cluster/apps, to review the loader job execution if necessary.



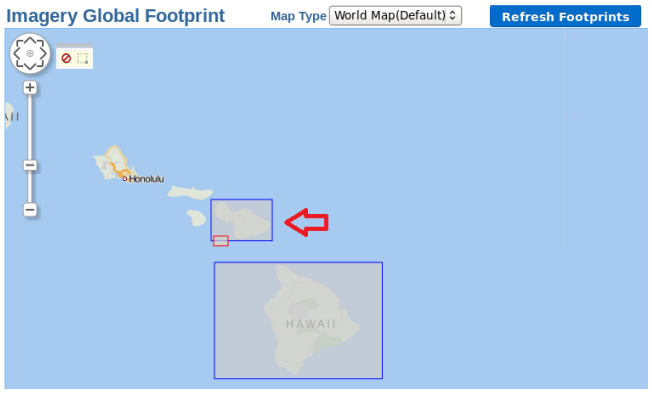
8. When done, go back to the imageserver, If no errors were generated, click on **Return**



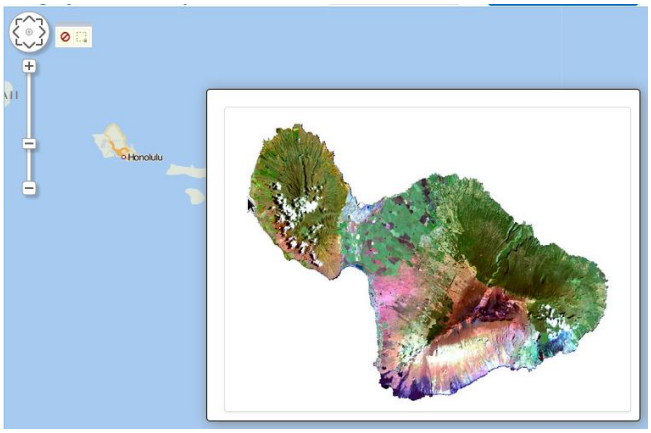
> ***Note**: if errors are generated in the loading process and download link will be provided to download the logs of this job.*

9. In Hadoop Loader and Globe tab, click on **Refresh Footprints**  button and wait till the footprints are drawn.

10. Click on each footprint to visualize the thumbnail of each image.

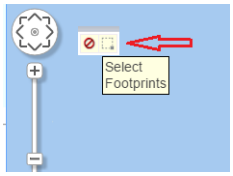


11. After clicking the rectangle the thumbnail appear, click on any part of the map to close it.

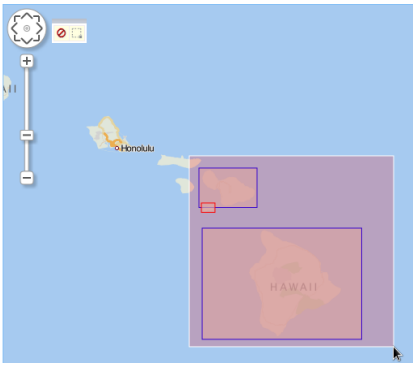


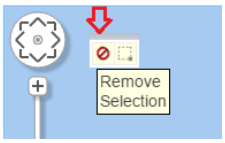
12. Zoom it or zoom out in the map as necessary to preview on the map the **footprints**.

13. Click on the selection mosaic area button at the left of the map, to start the **area selection:**

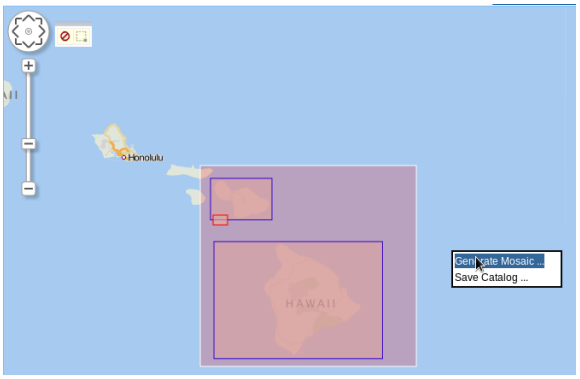


14. Then left-click on the map and **keep pressed** the button till the desired selection. This will create sub set operations on the rasters that **intersects** this selection.

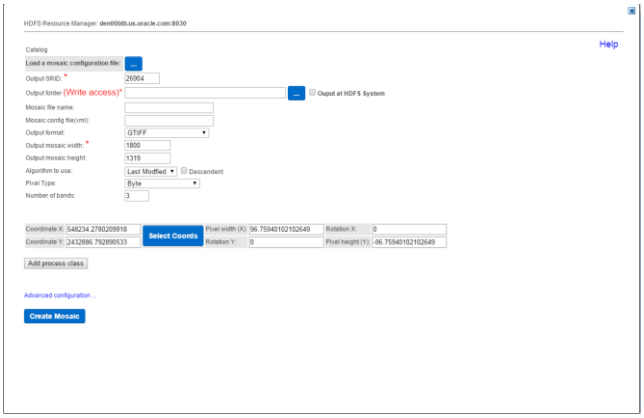


***Note**: to remove the selection just click on* 

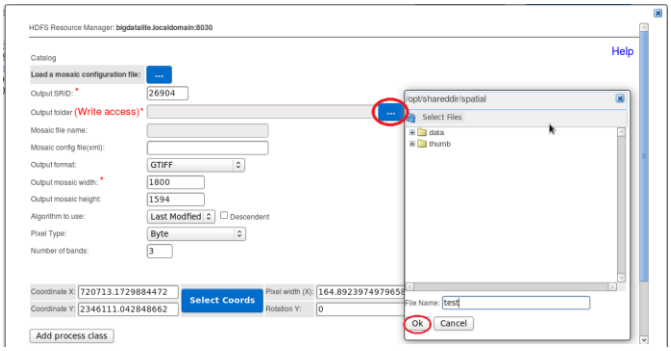
15. Right click on the map and then select **Generate Mosaic** submenu.



16. A mosaic form will be opened. **(Coordinates, size and SRID** will be generated automatically depending of the rasters selected).

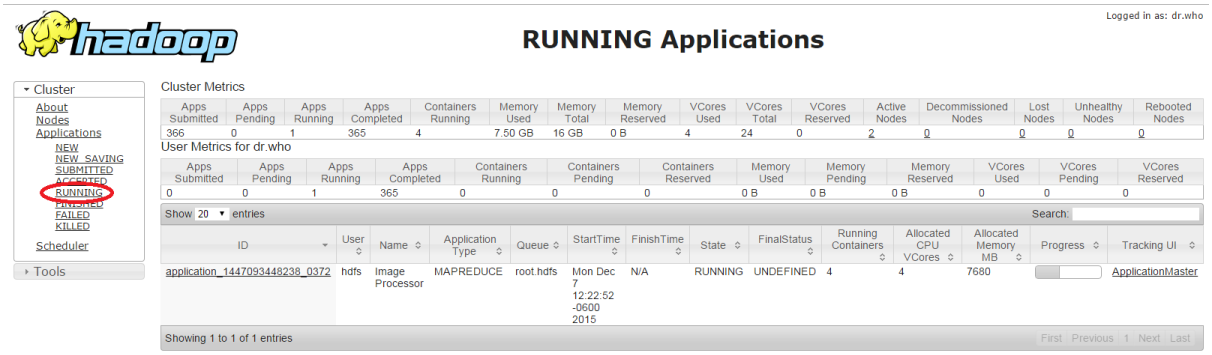


17. Provide an output mosaic name and output folder for the mosaic. Then click **ok**.



18. Click on Generate Mosaic Button at the end of the dialog, to start the mosaic and the subset operations. 

19. In the apps running jobs, refresh the page to see the job running http://localhost:8088/cluster/apps:



20. When done, return to the imageserver, and wait till the mosaic image is generated.

21. Scroll down to **Download** the mosaic 

. Or Go **back** to generate another mosaic with different configuration. 

> ***Hint**: Red rectangles out of the footprint box means that footprint has better resolution than the other in blue with which is intersecting, keep in mind this at the time of generating mosaic with the resolution algorithm*

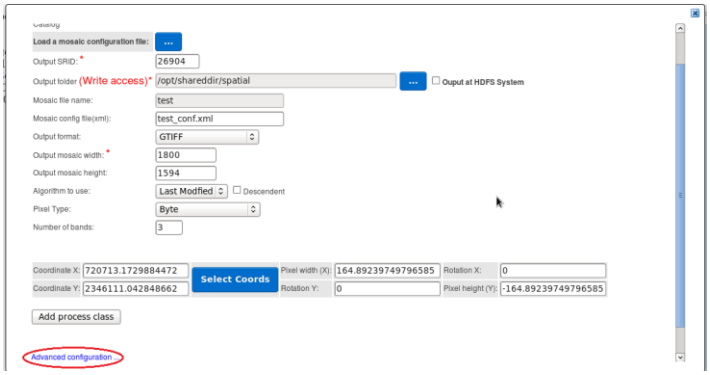
Generate mosaic with Spatial Operations.

In this section you will create a mosaic of rasters using the ImageServer Raster console with the raster you loaded previously. Please note that previous lab (Load raster images to HDFS and create basic

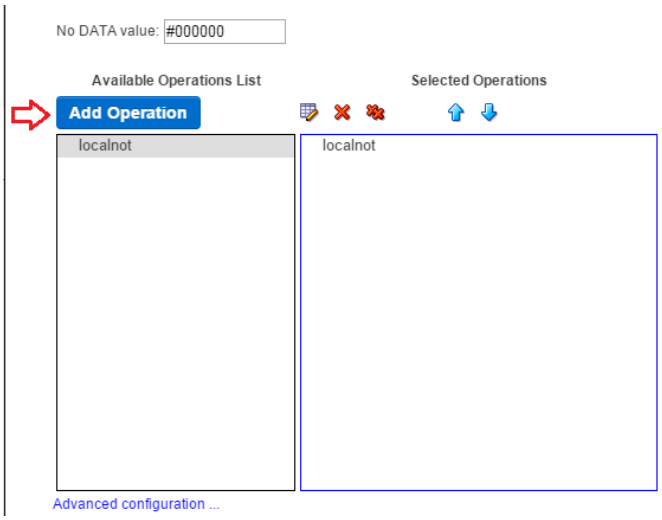
mosaic) is required to be executed before this one. The difference here is that you will add spatial raster operations to the process.

1. Press on 

or select a new area, over the mosaic dialog opened, click **on Advanced Configuration** section to open spatial operations.



2. From the left panel, select the localnot operation, and then click **add Operation** button to move it to the selected **operations panel**. Byte rasters only support localnot operation, but try different data types to list all supported raster operations.



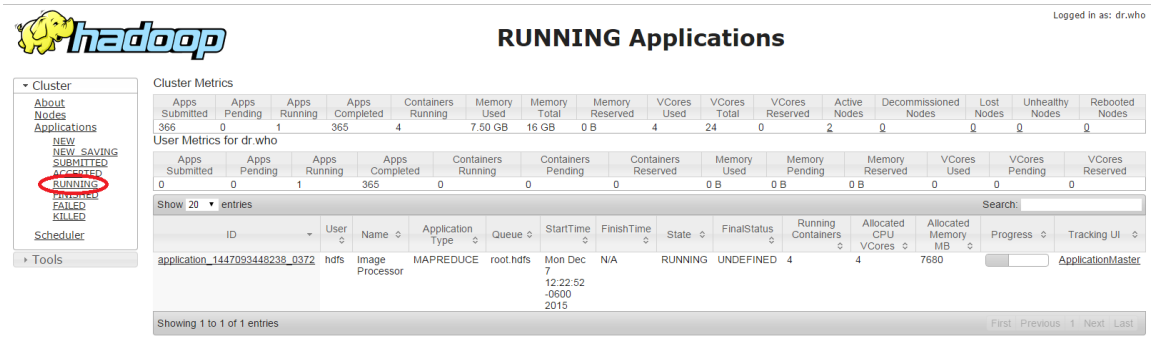
3. You may change the order of the selected operation by clicking on  when multiple operations are added.

> ***Note**: Available operations will be shown automatically depending of the pixel type of the images, not all of the operations are applicable to all varieties of pixel types.*

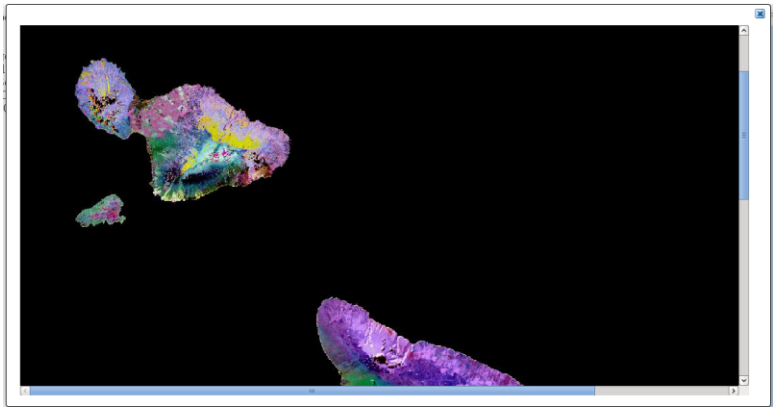
4. Click **Create Mosaic** button to generate a new mosaic. 

A warning about the output file already exists will appear, accept it since it will be overwritten.

5. Review the job process on the job running apps http://localhost:8088/cluster/apps.



6. In this mosaic we added a ***not*** operation that results in the inversion of every pixel value.



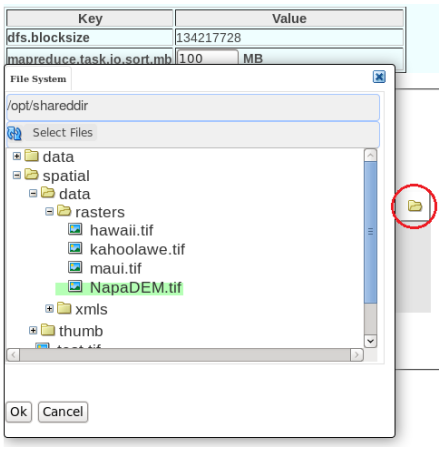
7. Download the mosaic and close the dialog. 

Load a DEM (Elevation Model) and calculate slope with algebra operation.

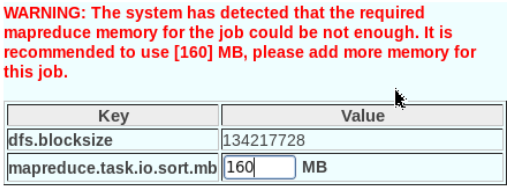
In this section we will load a special raster image capable to support a slope calculation and we will be able to visualize this image using a user-custom jar loaded into the image server. The imageserver API contains already a slope class to process the slope calculation.

1. Open http://localhost:8045/imageserver/#

2. Go to Hadoop **Loader and Globe** tab and load the **NapaDEM.tif** Image located at the root folder.

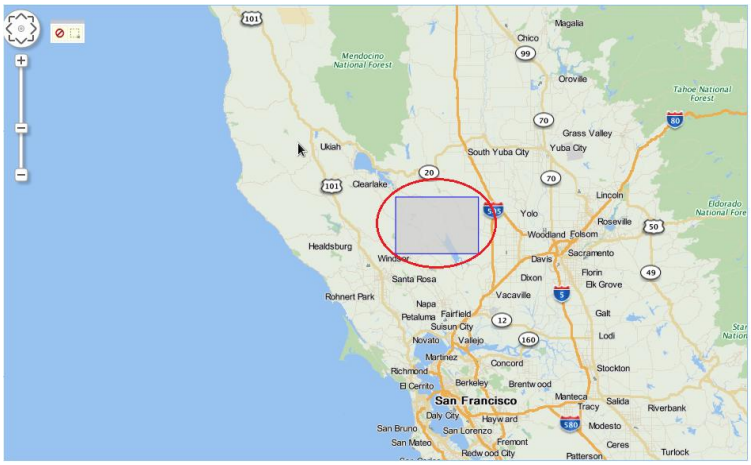


3. Provide the necessary memory to this job:

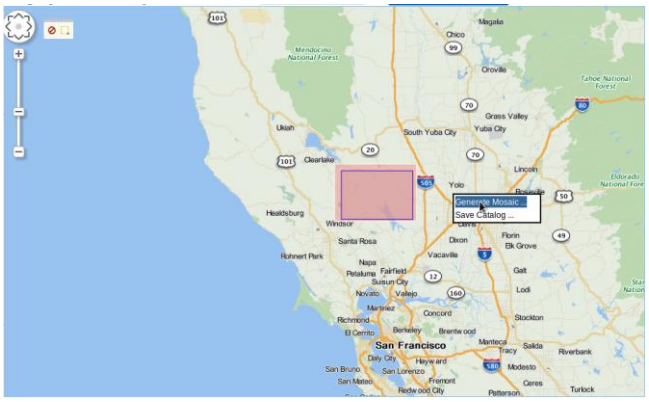


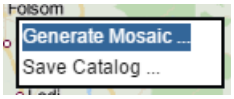
4. Click on Load Images button, and wait till the image it's loaded, when done, click on **Refresh Footprints** button located in the right panel. 

5. In the map you should be able to visualize the image around California.

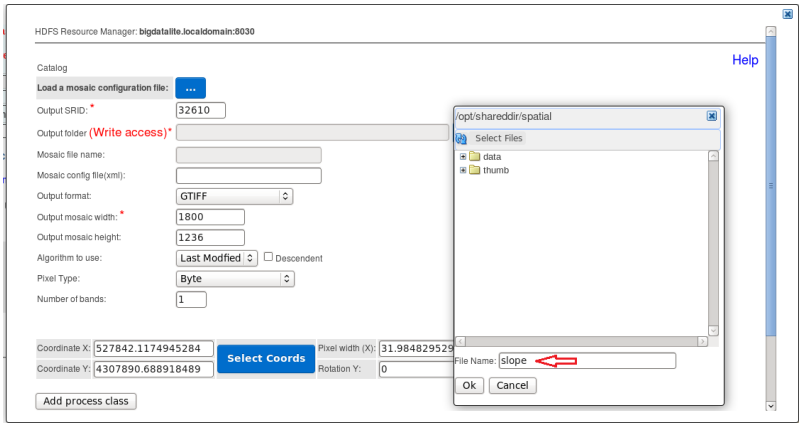


6. Click on the selection tool to select a mosaic area  , and select the entire footprint.

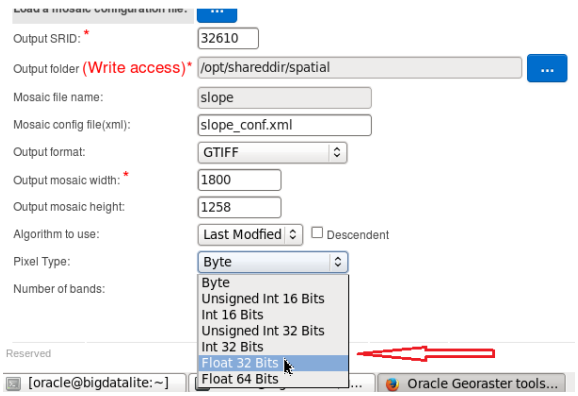


7. Then right click on the map to open the mosaic catalog configuration. 

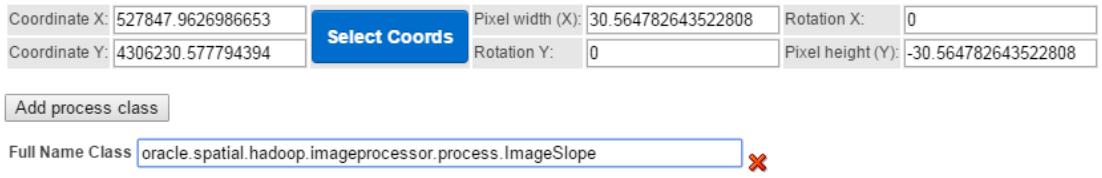
8. When the dialog is open, provide a mosaic output name for this image.



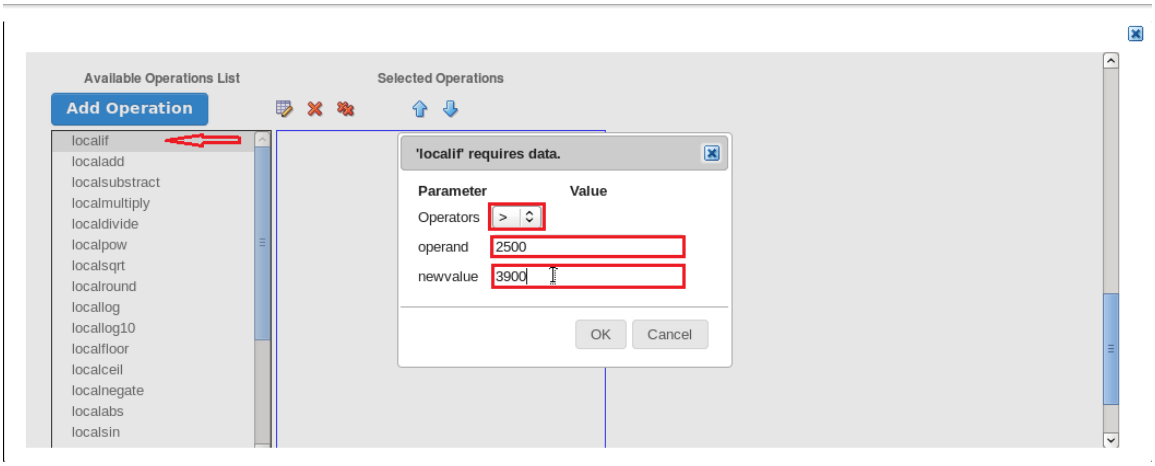
9. Modify the pixel type to Float 32 Bits since this is a 32 bits image.



10. Click on **Add process Class** button and provide the name of the class that will do the slope calculation(**fully class qualified name, including the package name**), in this case is: **oracle.spatial.hadoop.imageprocessor.process.ImageSlope**



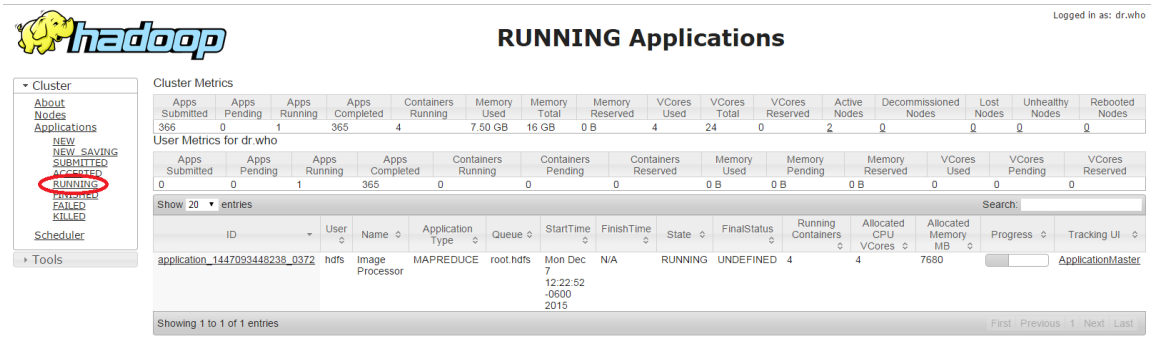
11. Add and algebra operation to modify the value of DEM pixel, in this case we will use the 'IF' operation to replace the value. Click *on Advance configuration and add the localif operation, and a popup dialog will appear to select the operator and the values.*



12. Click ok to save the configuration.

13. Click on 

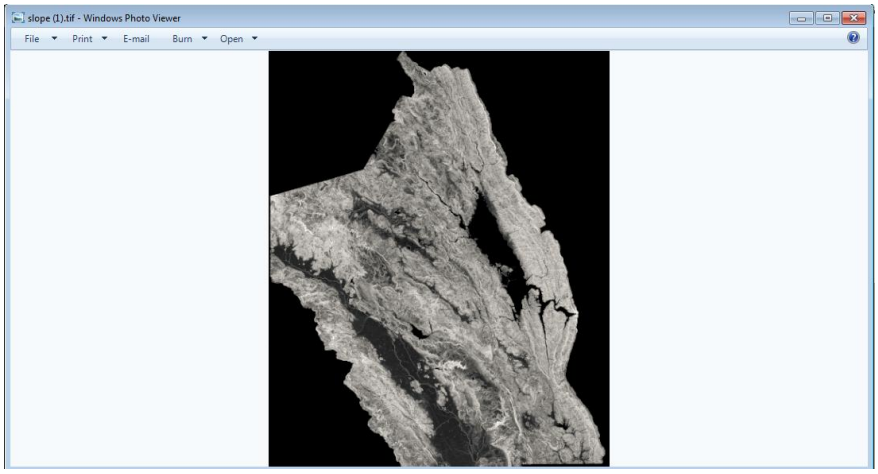
to generate mosaic and wait for the result. Review the job process on the job running apps http://localhost:8088/cluster/apps.



14. Once the process finished, go back to the imageserver tab and notice that the mosaic is shown, you won´t be able to visualize it in this viewer, if you wish to visualize it download the image

using the "Download mosaic" button and copy to a Windows machine.

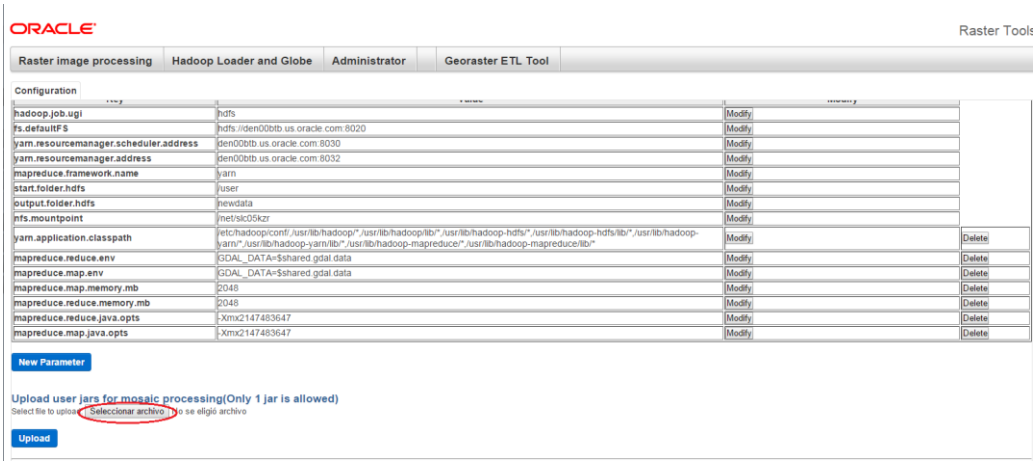
15. Open it using Windows Photo Viewer to see the slope.



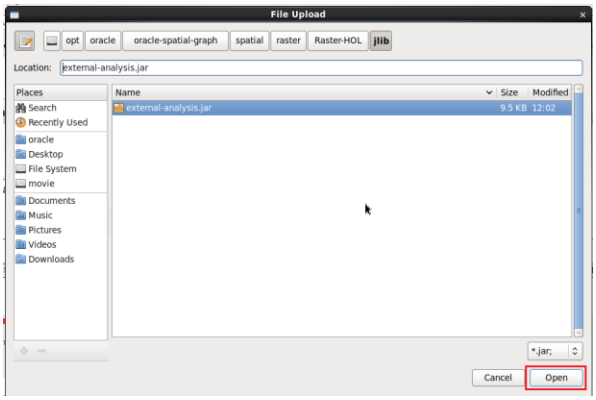
Calculate Hillshade on a DEM raster.

In this section we will load a special raster image capable to support a hillshade calculation and we will be using a external jar file with it, as an example that the framework can be extended and how to add new raster analysis support. This lab assumes you already loaded DEM into HDFS with the steps in previous lab.

1. Open http://localhost:8045/imageserver/# Go to **Administrator** tab and the section "Upload user jars for mosaic processing" at the bottom of the page, click on **Select File** button to select the external-analysis.jar located at */opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/jlib*

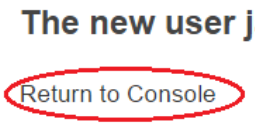


2. Select the user-custom jar and then click **open**.



3. Click on **upload**  button to add the custom jar to the imageserver.

4. Waits till the jar is loaded and shows a successful message, then click **Return to Console**.



5. Check the jar file is added to the **console**.

***Note**: If ERROR 404 appears it could be related to memory capacity overflow in the Virtual Machine when uploading the jar file, then you need to restart Jetty*

1). cd /u01/oracle-spatial-graph/spatial/jetty

2). ps –fea | grep java

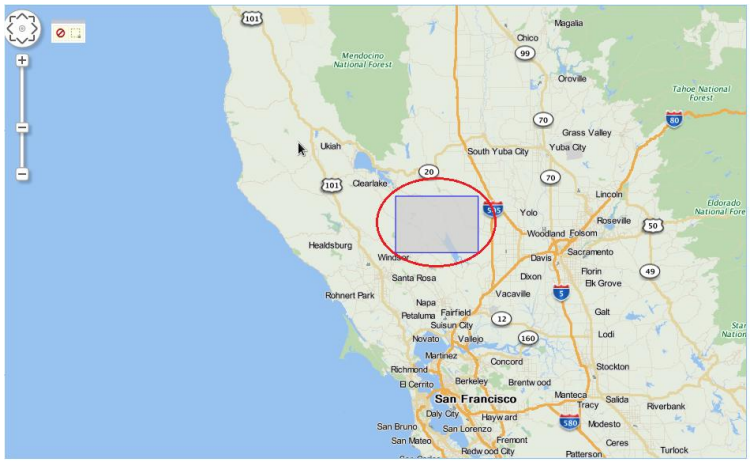
3). Select the <PID> belongs to the process java –jar start.jar

4). kill -9 <PID>

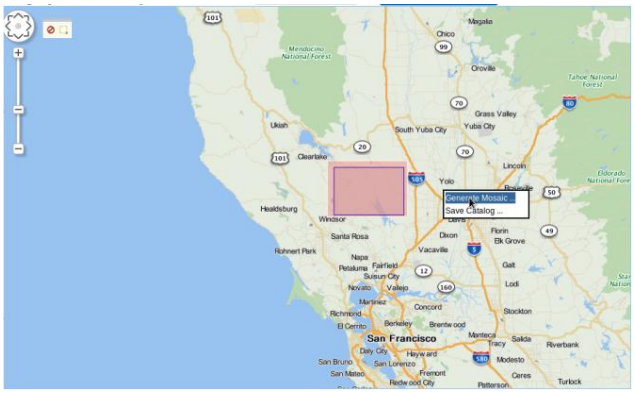
5) java –jar start.jar

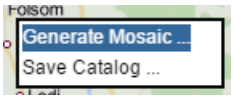
6. Go to the **Hadoop Loader and globe** tab 

and refresh the footprints to visualize the DEM image. 

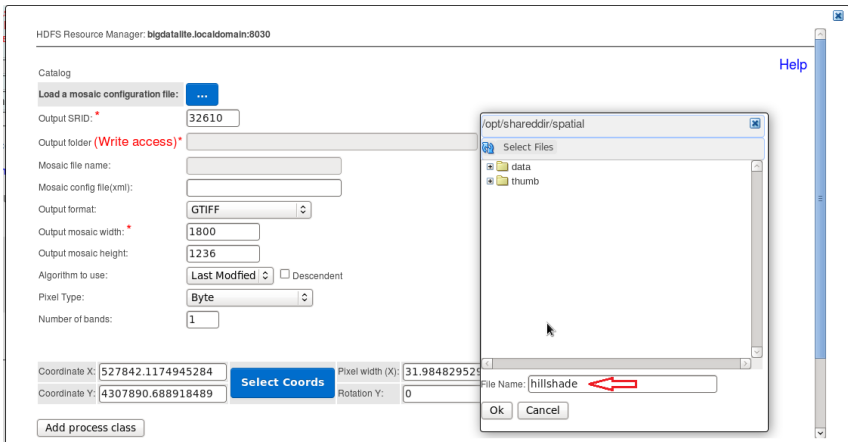


7. Click on the selection tool to select a mosaic area  , and select the entire footprint.

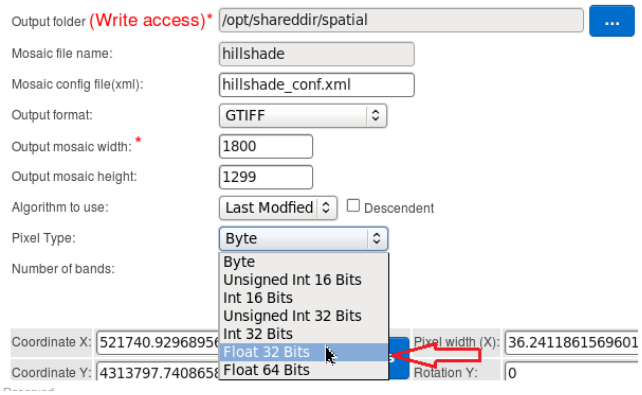


8. Then right click on the map to open the mosaic catalog configuration. 

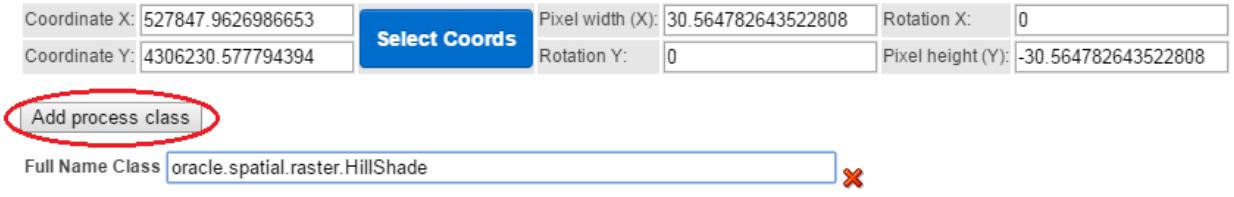
9. When the dialog is open, provide a mosaic output name for this image.



10. Modify the pixel type to Float 32 bits since this is a 32 bits image.

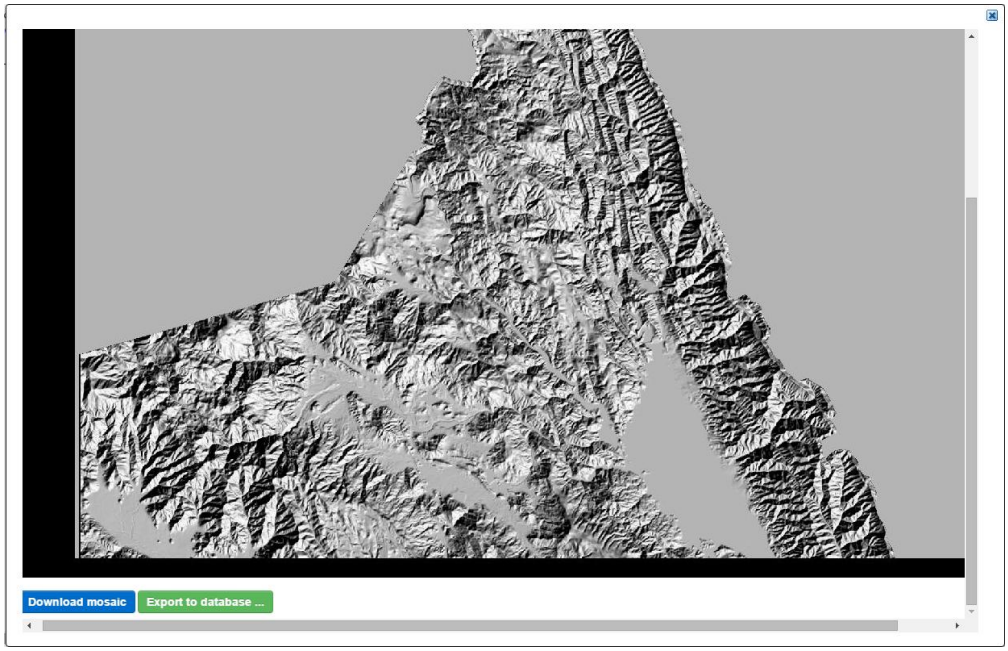


11. Click on **Add process Class** button and provide the name of the class that will do the slope calculation(**fully class qualified name, including the package name**), which is: **oracle.spatial.raster.HillShade**



12. Click on  to generate mosaic and wait for the result.

13. Download the hillshade image. **(This kind of image will not be visible with windows photo viewer, use another viewer)**



Lab Part 5: Use Oracle Big Data Spatial and Graph Raster Command Line

The following series of labs will walk us through the steps needed to load rasters in HDFS and process them using the Oracle Big Data Spatial and Graph Raster Command Line. First lab is a prerequisite for the rest of the labs since this loads the rasters into HDFS.

Lab Exercises:

1) Load a set of rasters (6 mins)
2) Process mosaic operation (6 mins)
3) Process mosaic subset operation and raster algebra functions (5 mins)
4) Process slope function (5 mins)
5) Extend the framework by creating a custom raster analysis operation (30 mins)

Let´s get started:

Step 1: Open the terminal.

Step 2: Make sure ***libproj.so*** native library under **/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib** has execute permissions for all users, if not, set them

```
chmod 755 /opt/oracle/oracle-spatial-
graph/spatial/raster/gdal/lib/libproj.so
```

Step 3: Make sure ***xml*** folder under **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/** has write permissions for all users, if not, set them

```
chmod 757 /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-
HOL/data/xml
```

Step 4: Switch your user id to ***hdfs*** user.

```
sudo su - hdfs
```

Step 5: Modify the **$HADOOP_CLASSPATH** environment variable to include all the jars required for job execution, by setting this, Hadoop will know where to find them.

```
export HADOOP_CLASSPATH=/opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/hadoop-imageprocessor.jar:/opt/oracle/oracle-
spatial-graph/spatial/raster/jlib/hadoop-raster-fwk-
api.jar:/opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/gdal.jar:$HADOOP_CLASSPATH
```

Load a set of rasters.

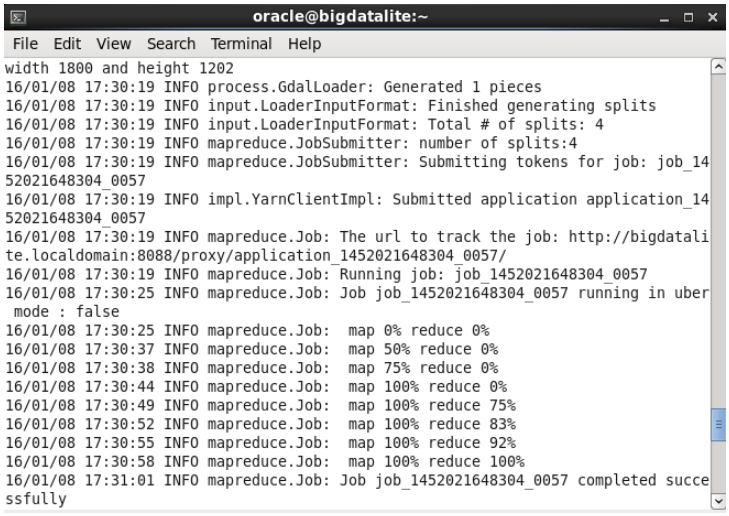
In these steps you will load in HDFS a set of rasters with different resolutions, data types and SRID´s.

1. Go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster** and review the rasters included there. There are three Hawaii rasters in different resolutions. There is also an Elevation Model of Northern Napa Valley in SRID 32610 and has a single. The four of them will be loaded in this exercise. The rasters to load are specified using the **–files** option when executing the job.

2. The rasters will be loaded in the HDFS folder you specify, for this exercise we are using ***exercise0*** folder, specifying the **–output** option when executing the job. This directory will be located in the following path**: /user/*hdfs*/exercise0/.** As you can notice, hdfs directory is included in the path, since that is the user executing the loading process.

3. You can also specify a thumbnail directory, where a thumbnail of every loaded raster is stored when the job finishes. You can use it to verify if the raster loaded correctly and no information was lost. For this exercise we are setting ***/opt/shareddir/spatial*** as the thumbnail folder, using the **–thumbnail** option when executing the job.
4. Another optional setting is the number of overlapping pixels, which indicate the pixels from the adjacent tiles that will be shared in every tile, these pixels apply to all directions (bottom, top, left and right). For this exercise we are setting this value as ***10,*** using the **–overlap** option when executing the job.
5. Gdal and Gdal data directories must be set using the **–gdal** and **–gdalData** options when executing the job, for this exercise we are setting **/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib** and **/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data**
6. Execute the job using the hadoop jar command with the **hadoop-imageloader.jar** and all the options described in the last steps and wait until it finishes.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageloader.jar -files /opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-
HOL/data/raster/hawaii.tif,/opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-
HOL/data/raster/kahoolawe.tif,/opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-
HOL/data/raster/maui.tif,/opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-HOL/data/raster/NapaDEM.tif    -out
exercise0     -overlap 10 -thumbnail /opt/shareddir/spatial   -gdal
/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib  -gdalData
/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data
```

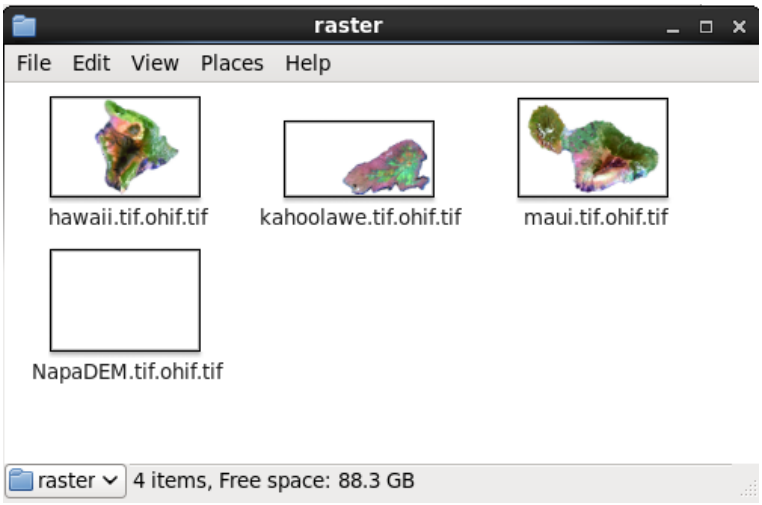


7. After the job finishes, list the content of the hdfs output directory to verify the rasters loaded correctly, the four rasters should be listed as a result.

```
hdfs dfs -ls /user/hdfs/exercise0/opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-HOL/data/raster
```

8. If you wish to visually check if the rasters were loaded correctly, the thumbnails are what you need. Review them in your **/opt/shareddir/spatial/thumb** directory followed by the hdfs path used in the previous step.

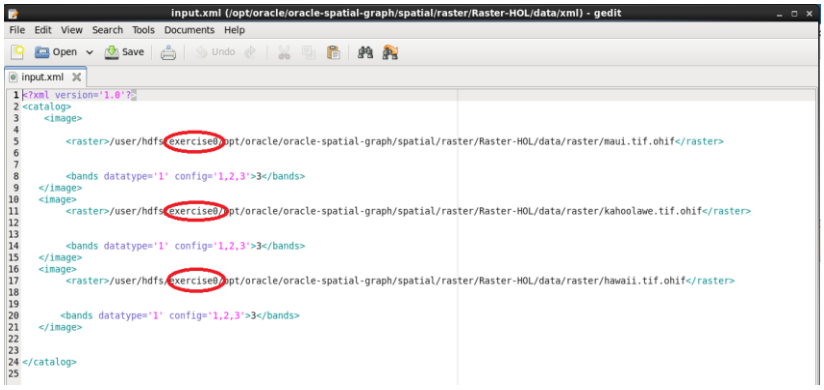
```
ls -lat
/opt/shareddir/spatial/thumb/user/hdfs/exercise0/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster
```



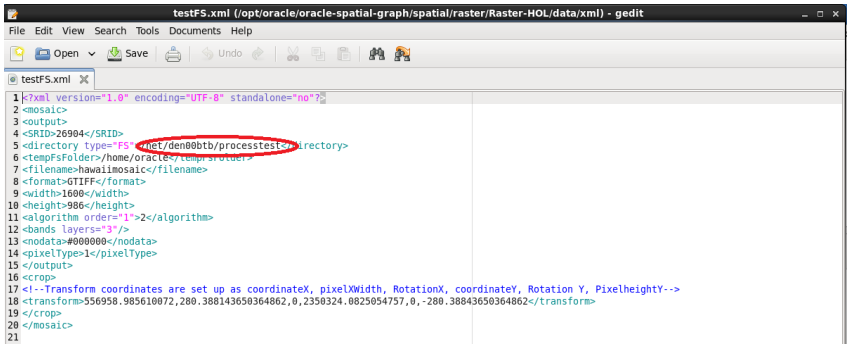
Process mosaic operation.

In these steps you will process the Hawaii rasters loaded in previous lab by extracting a subset of them and creating a mosaic with them (If you have not yet executed it, please do so before continuing).

1. Let´s start with the process by setting the raster catalog, go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster−HOL/data/xml** and open **input.xml**. Make sure the raster directory set in HDFS includes “exercise0” as part of the path for each raster in the catalog, they should all be in the path: **/user/hdfs/exercise0/opt/oracle/oracle-spatial-graph/spatial/raster/HOL/data/raster**. This file will be used when executing the job to set the catalog using the **–catalog** option.

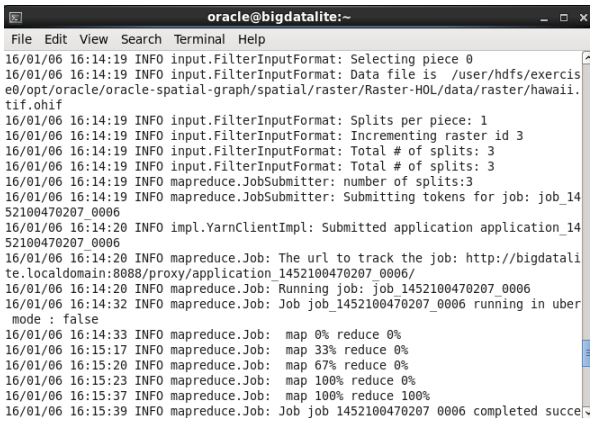


2. Next step is to verify the features of the mosaic we want to create, the mosaic configuration xml is **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testFS.xml**, in this example we are creating a mosaic that includes the complete three rasters according to the specified coordinates in the transform element, it has 3 bands, SRID 26904 and GTIFF format. Even though the raster kahoolawe listed in the catalog has SRID 26961, their coordinates will be transformed in the process to make them match the mosaic request of SRID 26904. The directory type element specifies that mosaic file output will be located in NFS, now you have to set the directory output to ***/opt/shareddir/spatial***. This file will be used when executing the job to set the mosaic configuration using the **–config** option. You may want to experiment by changing the coordinates and see how the resulting mosaic changes.

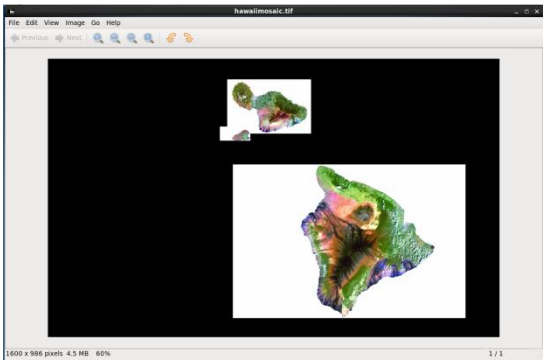


3. Now that the catalog and mosaic configuration are set, execute the job using the hadoop jar command with the **hadoop-imageprocessor.jar** including the options described in the last steps plus gdal paths and wait until it finishes.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-
imageprocessor.jar -catalog /opt/oracle/oracle-spatial-
graph/spatial/raster/Raster-HOL/data/xml/input.xml -config
/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-
HOL/data/xml/testFS.xml -gdal /opt/oracle/oracle-spatial-
graph/spatial/raster/gdal/lib  -gdalData /opt/oracle/oracle-spatial-
graph/spatial/raster/gdal/data
```



4. After the job finishes, go to **/opt/shareddir/spatial** and review that the mosaic was created there with the name **hawaiimosaic.tif**, you may visualize it and realize that all the three catalog rasters are together in this mosaic.



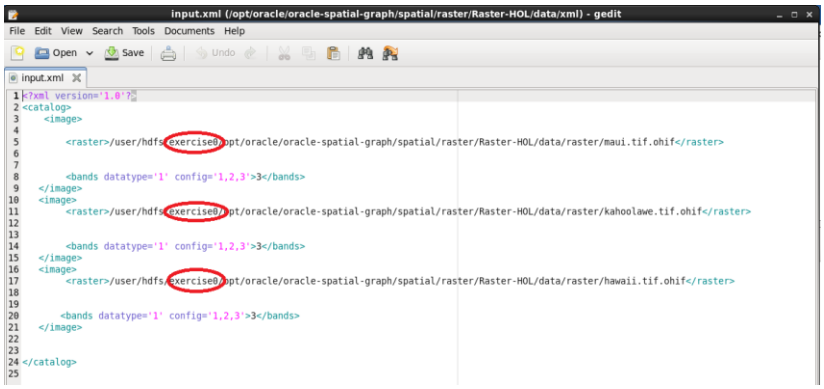
You may change the Black NODATA value in configuration xml, and you will see a different background color.

Process mosaic subset operation and raster algebra functions.

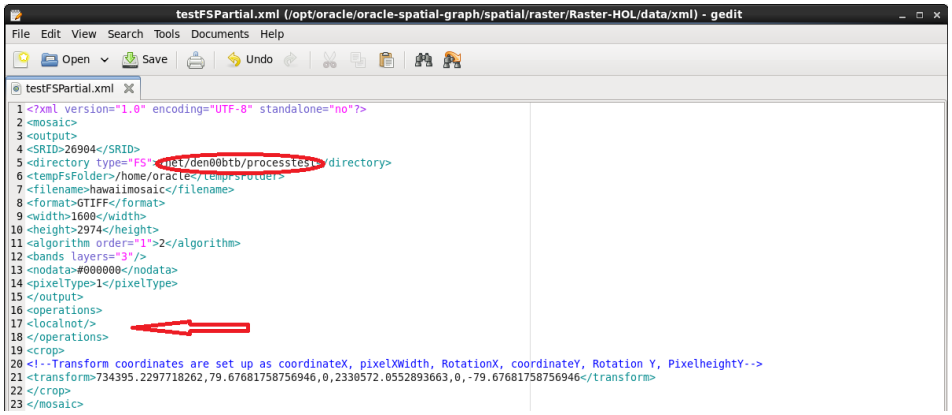
In these steps you will process the Hawaii rasters loaded previously in the first lab by extracting a subset of them(If you have not yet executed it, please do so before continuing), transform the pixel using a raster algebra function called ***localnot*** which inverts every bit in the pixels before creating a mosaic with them.

1. Let´s start with the process by setting the raster catalog, go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster−HOL/data/xml** and open **input.xml**. Make sure the raster directory set in HDFS includes “exercise0” as part of the path for each raster in the catalog, they should all be in the path: **/user/hdfs/exercise0/opt/oracle/oracle-spatial-graph/spatial/raster/HOL/data/raster**. The attribute config indicates the order of appearance

of the bands after the process is done, in this case is 1,2,3, you may want to change this order to experiment how the result changes. This file will be used when executing the job to set the catalog using the **–catalog** option.

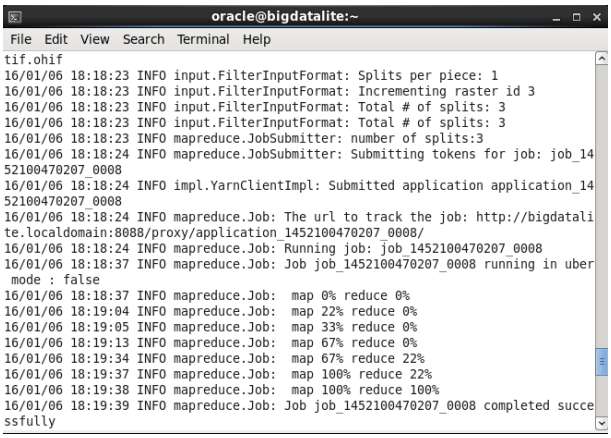


2. Next step is to verify the features of the mosaic we want to create, the mosaic configuration xml is **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testFSPartial.xml**, in this example we are creating a mosaic that includes the complete three rasters according to the specified coordinates in the transform element, it has 3 bands, SRID 26904 and GTIFF format. The directory type element specifies that mosaic file output will be located in NFS, now you have to set the directory output to ***/opt/shareddir/spatial***. This file will be used when executing the job to set the mosaic configuration using the **–config** option. If you notice, the transform coordinates have changed from the last exercise, now the mosaic has a minor pixelWidth and pixelHeight , which will cut off the last island on the right, which is hawaii.tif raster, you will visualize this on the mosaic output one is processed. There is a new element in this xml, <operations>, where localnot operation is defined, this operation inverts the bits in every pixel and is part of the supported raster algebra operations included in the framework.



3. Now that the catalog and mosaic configuration are set, execute the job using the hadoop jar command with the **hadoop-imageprocessor.jar** and all the options described in the last steps and wait until it finishes.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-imageprocessor.jar -catalog /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/input.xml -config /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testFSPartial.xml -gdal /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib -gdalData /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data
```

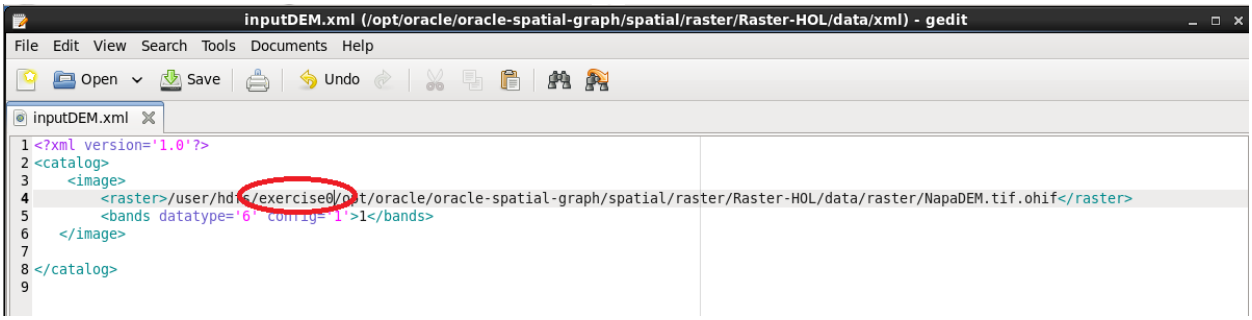


4. After the job finishes, go to **/opt/shareddir/spatial** and review that the mosaic was created there, you may visualize it and realize that all the three catalog rasters are together in this mosaic. In this example the last island on the right is cut, this is a result of the transform coordinates set in the mosaic configuration, that do not cover the complete island, and is an example of subset operation. Also, you may notice the change in the pixels color and this is the result of applying the *localnot* operation.

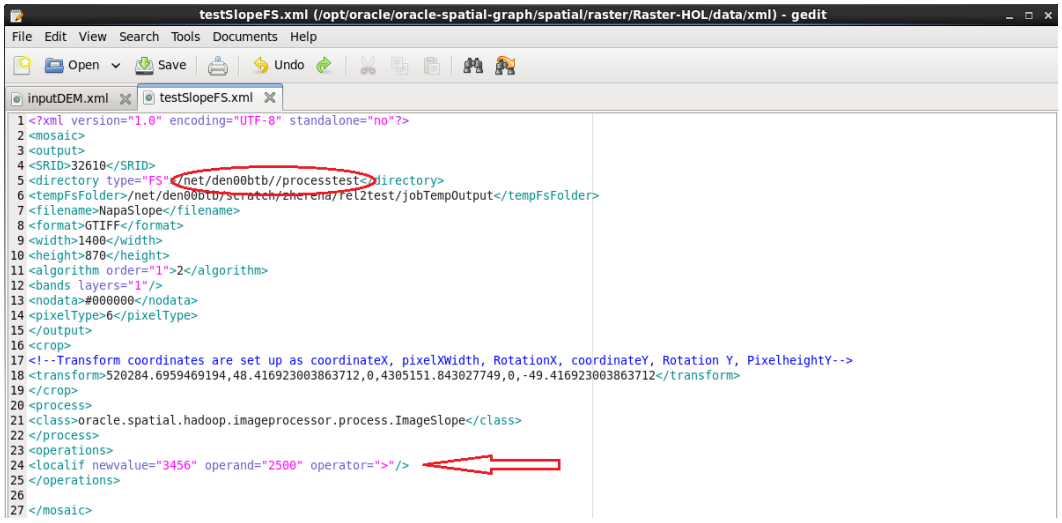
Process slope function.

In these steps you will process the Napa Valley Elevation Model loaded in HDFS in the first lab (If you have not yet executed it, please do so before continuing) using a slope function provided by the framework.

1. Let´s start with the process by setting the raster catalog, go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml** and open **inputDEM.xml**. Make sure the raster directory contains "exercise0" as part of the path for the raster in the catalog, it should be in the path: **/user/hdfs/exercise0/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster**. This file will be used when executing the job to set the catalog using the **–catalog** option.

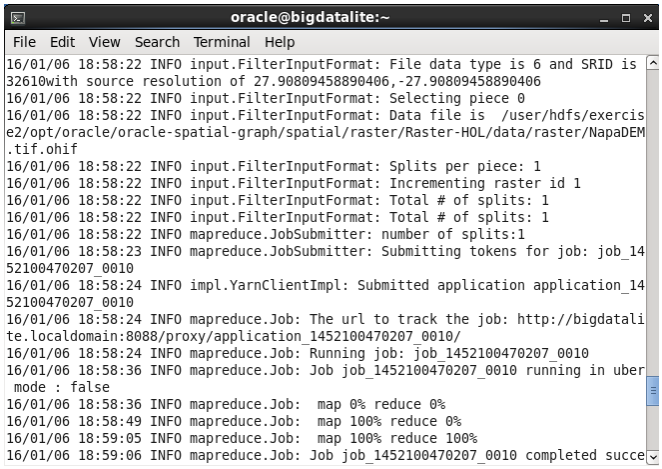


2. Next step is to verify the features of the mosaic we want to create, the mosaic configuration xml is **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testSlopeFS.xml**, in this example we are creating a mosaic that include the complete Elevation Model according to the specified coordinates in the transform element, it has 1 band, SRID 32610 and GTIFF format. The directory type element specifies that mosaic file output will be located in NFS, now you have to set the directory output to ***/opt/shareddir/spatial***. The <operations> element sets the *localif* operation, this operation updates the value of every pixel where its original elevation is greater than 2500, you may change this value if you wish to experiment with it. The <process> element indicates that ImageSlope class, which is part of the framework will be executed as part of the process, as a result the output mosaic will show the slope according to its neighbor pixels. This file will be used when executing the job to set the mosaic configuration using the **–config** option.

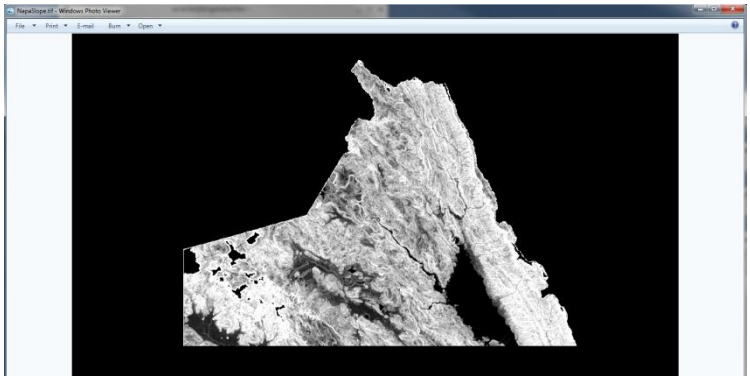


3. Now that the catalog and mosaic configuration are set, execute the job using the hadoop jar command with the **hadoop-imageprocessor.jar** and all the options described in the last steps and wait until it finishes.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-imageprocessor.jar -catalog /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/inputDEM.xml -config /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testSlopeFS.xml -gdal /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib -gdalData /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data
```



4. After the job finishes, go to **/opt/shareddir/spatial** and review that the mosaic was created there, the output raster with the slope cannot be visualized in Linux environment for the standard image tools, if you want to do so you may want to open the raster in Windows environment with Windows Photo Viewer and realize that the original Elevation Model has changed, now it shows the slope of every pixel allowing you to have a mountain view. If you don't have access to Windows environment you may try any specialized raster tool.

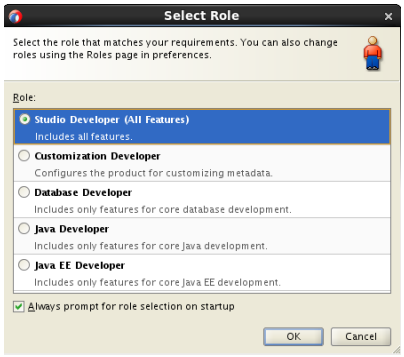


Extend the framework by creating a custom raster analysis operation.

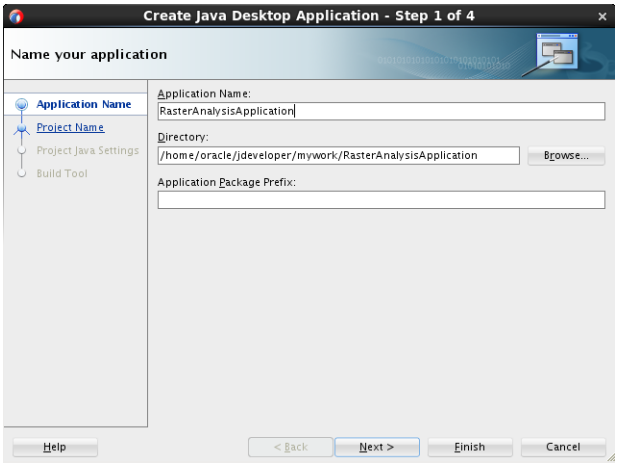
In these steps you will learn how to extend the framework to create your own custom raster analysis class. We will create a hillshade operation and will plug it to the framework to use it. The Hillshade function obtains the hypothetical illumination of a surface by determining illumination values for each pixel in a raster. It does this by setting a position for a hypothetical light source and calculating the illumination values of each pixel in relation to neighboring pixels. You will process it using your own processing class created by you! The input raster is the Napa Valley Elevation Model loaded in HDFS in the first lab (If you have not yet executed it, please do so before continuing).

1. Let´s create the hillshade processing class. Open JDeveloper by clicking the JDev icon in the upper Task Bar.

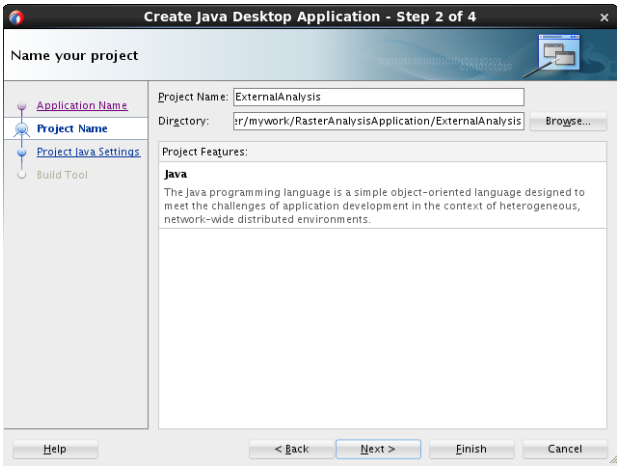
2. Select the Studio Developer role, and you may want to uncheck the “Always prompt for role selection on startup” checkbox so this window does not appear next time.



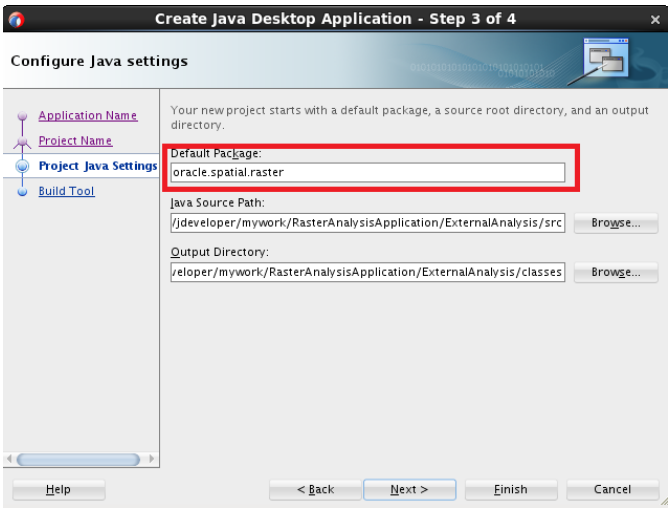
3. Go to Application Navigator and from the dropdown list select New Application, select Java Desktop Application and click OK. Set the Application Name to *RasterAnalysisApplication* and click Next.



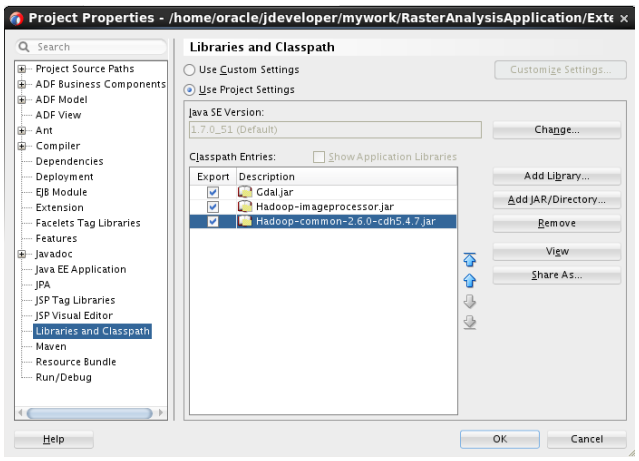
4. Set the project name to ExternalAnalysis and click Next.



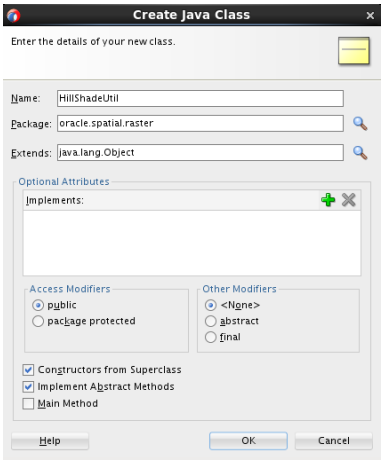
5. Set the default package to oracle.spatial.raster and click Finish.



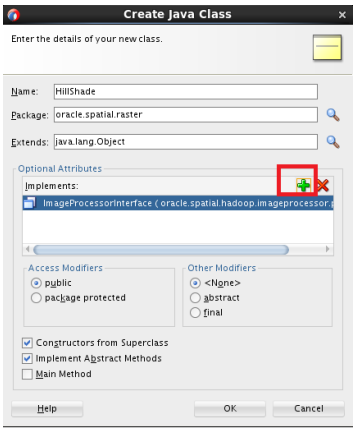
6. Right click on the ExternalAnalysis project in the Application Navigator and select Project Properties. Select the Libraries and Classpath option and click on Add JAR/Directory. In the Add Archive/Directory window set the path to **/opt/oracle/oracle-spatial-graph/spatial/raster/jlib** and select the jar files **gdal.jar** and **hadoop-imageprocessor.jar**. Click Open and the Ok. From the **/usr/lib/hadoop** add the **hadoop-common-2.6.0-cdh5.4.7.jar**. Now the project knows about the required files to create the hillshade class.



7. Right click on the ExternalAnalysis project in the Application Navigator and select New -> Java Class. In the Create Java Class window, set *HillShadeUtil* as the Name for the class and click Ok.



8. Go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/java/src** and copy the content of *HillShadeUtil.java* into the class you just created in JDeveloper. This class contains utilitary methods to calculate hillshade value for every pixel in the DEM.
9. Right click on the ExternalAnalysis project in the Application Navigator and select New Java Class. In the Create Java Class window, set *HillShade* as the Name for the class. Click on the green Add button to add the class we are implementing which is *oracle.spatial.hadoop.imageprocessor.process.ImageProcessorInterface*, then click Ok.



10. Let´s start programming the HillShade.java class; The following import statements are required:

```
import oracle.spatial.hadoop.imageprocessor.datatype.ImageBandWritable;
import oracle.spatial.hadoop.imageprocessor.process.ImageProcessorInterface;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.gdal.gdal.gdal;
```

11. Since the objective of this lab is to understand how to extend the Oracle framework using the ImageBandWritable datatype that contains the raster info, all the hillshade algorithm details are separated in the *HillShadeUtil* class so that the class you are coding focus only in getting info from the raster and set the process results back in the ImageBandWritable data type. If you are interested in the algorithm for hillshade, review the utilitary class. Inside the *process* method that created automatically to implement the ImageProcessorInterface, we will start developing the hillshade calculation, the following code gets the bands number, width and height of the tile, number of overlapping pixels and flags for overlap in the four borders of every pixel.

```
//the GeneralInfoWritable inside the image contains the information about the final mosaic output, get number of output bands from
there.
int bands = Integer.valueOf(imageBandWritable.getMosaicInfo().getBandsToAdd().toString().split(",").length);

int width = imageBandWritable.getDstWidthSize().get();
int height = imageBandWritable.getDstHeightSize().get();

int overlap = imageBandWritable.getOverlap().get();
byte upOv = imageBandWritable.getUpOv().get();
byte downOv = imageBandWritable.getDownOv().get();
byte leftOv = imageBandWritable.getLeftOv().get();
byte rightOv = imageBandWritable.getRightOv().get();
```

12. Now let´s define the MBR where the hillshade algorithm will start the processing, an initial setup of 0 as a start and width and height of the tile as end is done. As a second step, we validate the flags for overlapping pixels in every border of the tile and adjust the MBR accordingly to make sure that only pixels specific to this tile are processed and avoid redundant processing with other mappers, also the offsets in X and Y for this tile are updated based on the overlapping pixels.

```
int startX = 0;
int startY = 0;
int endX = width;
int endY = height;
//If there are overlapping pixels on the left, then X will start in position overlap otherwise in 1
if (leftOv == Byte.MAX_VALUE) {
        startX = overlap;
        imageBandWritable.setOffX(new IntWritable(imageBandWritable.getOffX().get() + overlap));
} else {
        startX = 1;
        imageBandWritable.setOffX(new IntWritable(imageBandWritable.getOffX().get() + 1));
}
//If there are overlapping pixels on the right, then X will end in the width position minus overlapping pixels
if (rightOv == Byte.MAX_VALUE) {
        endX = width - overlap;
}
//If there are overlapping pixels on the top, then Y will start in position overlap
if (upOv == Byte.MAX_VALUE) {
        startY = overlap;
        imageBandWritable.setOffY(new IntWritable(imageBandWritable.getOffY().get() + overlap));
}
//If there are overlapping pixels on the bottom, then Y will end in the height position minus overlapping pixels
if (downOv == Byte.MAX_VALUE) {
        endY = height - overlap;
}
```

13. Now with the extracted information, let´s call the hillshadeTile method and finally return the ImageBandWritable object with the hillshade information for final mosaic processing inside the framework.

```
hillshadeTile(imageBandWritable, startX, startY, endX, endY, width, height, bands);
 return imageBandWritable;
```

14. Let´s create the hillshadeTile method that returns void and receives the data we extracted. The first statement will be the creation of a HillShadeUtil instance. The rest of the method is described in the next items.

```
private void hillshadeTile(ImageBandWritable image, int startX, int startY, int endX, int endY, int width,
                                int height, int bands) {
        HillShadeUtil util = new HillShadeUtil();

        //More code to add inside this method
}
```

15. Now calculate the last position in the tile by multiplying width*height, extract the width for pixels in X and Y from mosaic configuration object, and calculate the width and height of the piece to process according to the start and end pixels position excluding the overlapping pixels that we calculated in last code segment. Reset the band counter of the ImageBandWritable instance, so that we can add new processed data to it starting on band index 0.

```
int lastValue = width * height;
double pixelXWidth = image.getMosaicInfo().getPixelXWidth().get();
double pixelYWidth = image.getMosaicInfo().getPixelYWidth().get();
int newWidth = endX - startX;
int newHeight = endY - startY;
```

```
image.resetBandCounter();
```

16. We have all the required information to calculate hillshade, let´s iterate the bands, get the NODATA value for each of them and execute the *util.hillshade* function. If you have interest on the hillshade algorithm please review HillShadeUtil class.

```
for (int x = 1; x <= bands; x++) {
        DoubleWritable noData = image.getNoDataArray(x);
        util.hillshade(lastValue, pixelXWidth, pixelYWidth, noData, image, newWidth, newHeight, x, startX,
                startY, endX, endY, width, height);
}
```

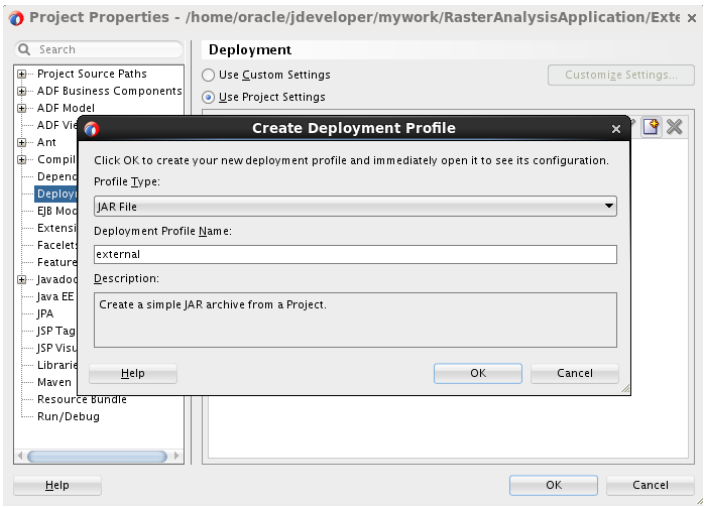
17. Now that hillshade has been calculated and set in the ImageBandWritable instance, let´s set the final details to it, that reflect the new MBR for this tile, starting by setting the width, height and number of bytes.

```
image.setDstWidthSize(new IntWritable(newWidth));
image.setDstHeightSize(new IntWritable(newHeight));
int pixels = (newWidth) * (newHeight);
int bytesNumber = pixels * (gdal.GetDataTypeSize(image.getDType().get()) / 8);
image.setBytesNumber(bytesNumber);
```

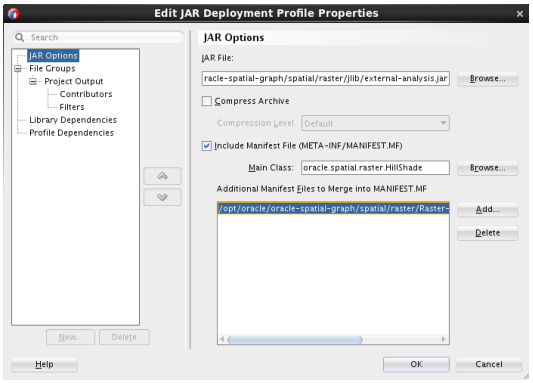
18. Finally, update the coordinates for this tile, first get the original coordinates of the tile, and then update positions 0 and third of the substitute array since those represent the raster start point, update it by multiplying the pixel width and height in positions 1 and 5 * the new start X and Y pixels, this gives us the delta for the new MBR and must be added to original start coordinates.

```
DoubleWritable[] originalGeoTransform = image.getGeoTransform();
DoubleWritable[] adfGeoTransform = new DoubleWritable[6];
//0 and 3rd positions represent image origin, recalculating for new image size.
adfGeoTransform[0] =
   new DoubleWritable(originalGeoTransform[0].get() + (originalGeoTransform[1].get() * startX));
adfGeoTransform[1] = new DoubleWritable(originalGeoTransform[1].get());
adfGeoTransform[2] = new DoubleWritable(originalGeoTransform[2].get());
adfGeoTransform[3] =
   new DoubleWritable(originalGeoTransform[3].get() + (originalGeoTransform[5].get() * startY));
adfGeoTransform[4] = new DoubleWritable(originalGeoTransform[4].get());
adfGeoTransform[5] = new DoubleWritable(originalGeoTransform[5].get());
image.setGeoTransform(adfGeoTransform);
```

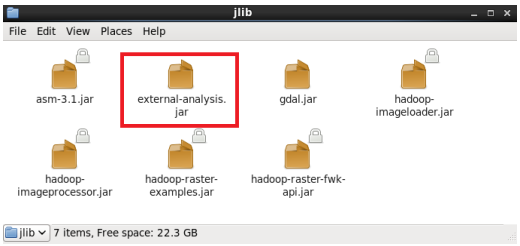
19. Class is ready! Now let´s build it and pack a jar file to execute it. Right click on the project and select Project Properties. In the emergent window select Deployment option and click on Create Deployment Profile icon. Select JAR File as the Profile Type and set external as the Deployment Profile Name, click OK to Finish.



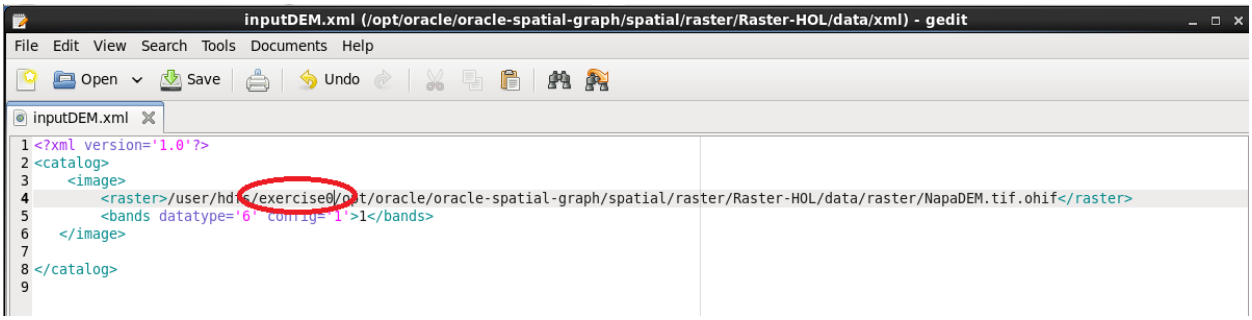
20. In the emergent Edit JAR Deployment Profile Properties, click the Browse button to set the main class to oracle.spatial.raster.HillShade and click Ok. Set the JAR File output path to **/opt/oracle/oracle-spatial-graph/spatial/raster/jlib**, the name to external-analysis.jar, add an additional MANIFEST file by clicking the button Add, and select the MANIFEST-EXTERNAL file included in **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/java/** and click Ok to accept all changes.



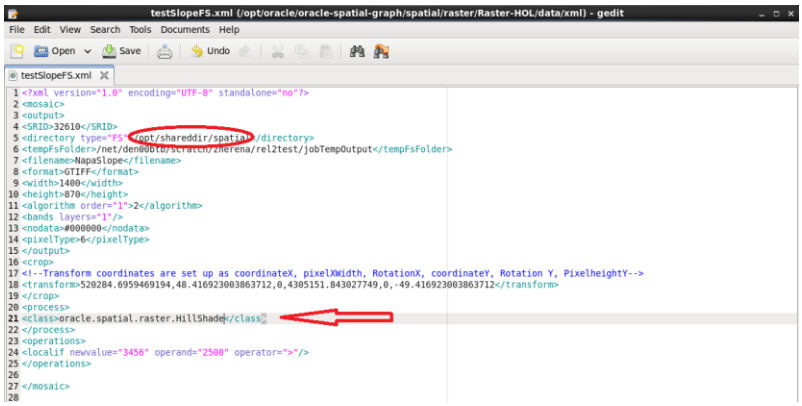
21. Deployment profile is ready, build the project using <Ctrl> + <F9>, after it is built, right click on the project select Deploy and the external profile you just created and click OK. Jar file is deployed and ready to use. This jar file has already been exported to HADOOP_CLASSPATH in the Preparation Steps located at the beginning of this lab part.



22. Now configure the raster catalog to process, go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml** and open **inputDEM.xml**. Make sure the raster directory set in HDFS (exercise0) is part of the path for the raster in the catalog, it should be in the path: **/user/hdfs/exercise0/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster**. This file will be used when executing the job to set the catalog using the **–catalog** option.

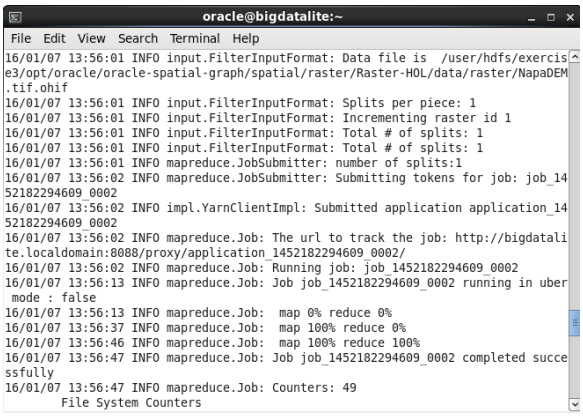


23. Next step is to verify the features of the mosaic we want to create, the mosaic configuration xml is **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testSlopeFS.xml**, in this example we are creating a mosaic that include the complete Elevation Model according to the specified coordinates in the transform element, it has 1 band, SRID 32610 and GTIFF format. The directory type element specifies that mosaic file output will be located in NFS, now you have to set the directory output to **/opt/shareddir/spatial**. The <operations> element sets the *localif* operation, this operation updates the value of every pixel where its original elevation is greater than 2500, you may change this value if you wish to experiment with it. The <process> element indicates the class that will be used to process the raster, set oracle.spatial.raster.HillShade, which is the class you just built and as a result the output mosaic will show the hillshade of the elevation model. You may also change the output filename. The class will be found by Hadoop since the jar file was already exported to HADOOP_CLASSPATH on step 8. This file will be used when executing the job to set the mosaic configuration using the **–config** option.

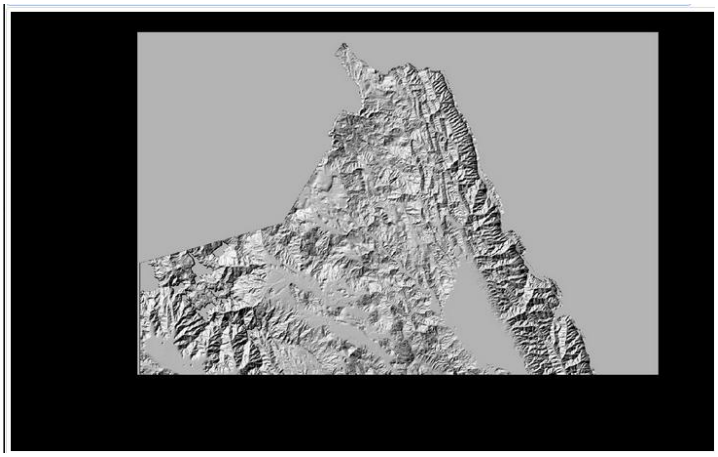


24. Use **–usrlib** option in the command line to set the external-analysis.jar file that you just created and share it to all nodes in the cluster
25. Now that the catalog and mosaic configuration are set, execute the job using the hadoop jar command with the **hadoop-imageprocessor.jar** and all the options described in the last steps and wait until it finishes.

```
hadoop jar /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/hadoop-imageprocessor.jar -catalog /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/inputDEM.xml -config /opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/xml/testSlopeFS.xml -gdal /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib -gdalData /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data -usrlib /opt/oracle/oracle-spatial-graph/spatial/raster/jlib/external-analysis.jar
```



26. After the job finishes, go to **/opt/shareddir/spatial** and review that the mosaic was created there, the output raster with the hillshade cannot be visualized in Linux environment for the standard image tools, if you want to do so you may want to open it by adding it to a raster catalog in ImageServer (1. http://localhost:8045/imageserver/#. 2. Raster Image Processing -> Catalog -> New Catalog -> Raster Catalog. 3. Imagery -> Add File Image -> /opt/shareddir/spatial/hillshadefile.tif -> Ok).



Lab Part 6: Create Customized Jobs Using the Oracle Big Data Spatial and Graph Raster API

The following series of labs will walk us through the steps needed create MapReduce jobs that load rasters in HDFS and process them using the Oracle Big Data Spatial and Graph Raster API.

Lab Exercises:

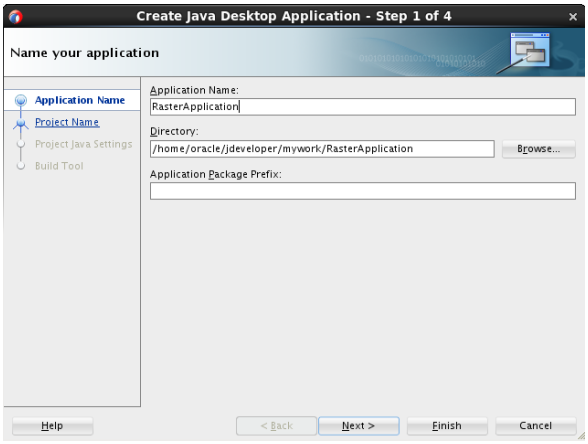
Load a set of rasters and process the mosaic operation (18 mins)
Load a DEM and process slope function (10 mins)

Let´s get started:

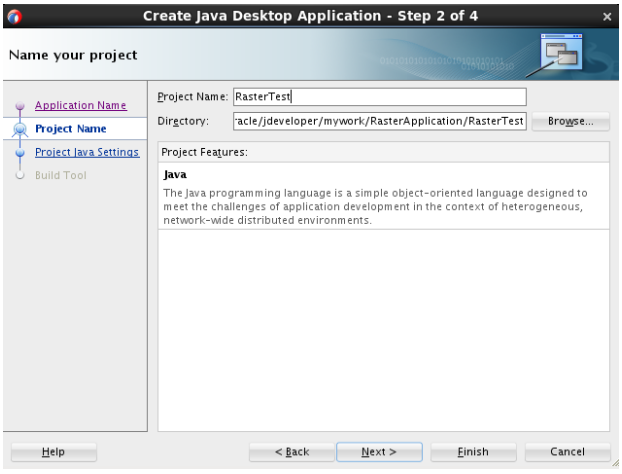
Step 1: Open JDeveloper by clicking the JDev icon in the upper Task Bar (7 minutes for the following steps).

Step 2: Select the Studio Developer role, and you may want to uncheck the “Always prompt for role selection on startup” checkbox so this window does not appear next time.

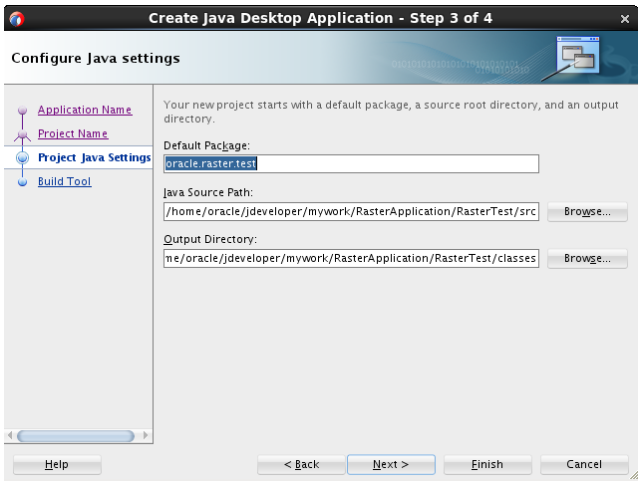
Step 3: Go to Application Navigator and from the dropdown list select New Application, select Java Desktop Application and click OK. Go to Application Navigator and from the dropdown list select New Application. Set the Application Name to *RasterApplication* and click Next.



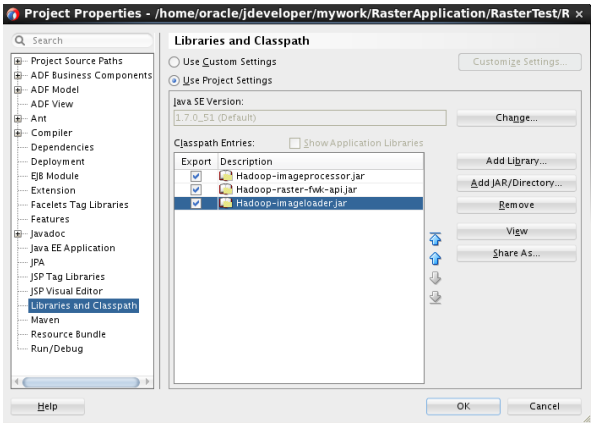
Step 4: Set the project name to RasterTest and click Next.



Step 5: Set the default package to oracle.raster.test and click Finish.



Step 6: Right click on the RasterTest project in the Application Navigator and select Project Properties. Select the Libraries and Classpath option and click on Add JAR/Directory. In the Add Archive/Directory window set the path to **/opt/oracle/oracle-spatial-graph/spatial/raster/jlib** and select the jar files hadoop-raster-fwk-api.jar, hadoop-imageloader.jar and hadoop-imageprocessor.jar. Click Open and the Ok. Now the project knows about the required files for raster processing.



Step 7: Open the terminal.

Step 8: Make sure libproj.so native library under /opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib has execute permissions for all users, if not, set them

```
chmod 755 /opt/oracle/oracle-spatial-
graph/spatial/raster/gdal/lib/libproj.so
```

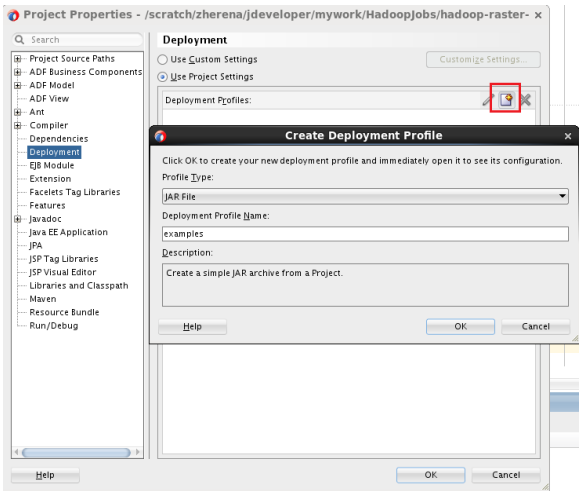
Step 9: Switch your user id to *hdfs* user.

```
sudo su - hdfs
```

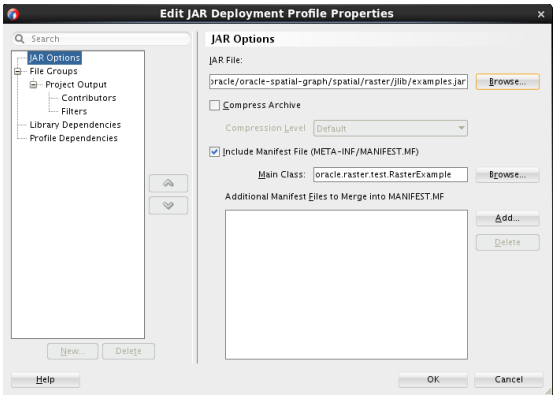
Step 10: Modify the $HADOOP_CLASSPATH environment variable to include all the jars required for job execution, by setting this, Hadoop will know where to find them.

```
export HADOOP_CLASSPATH=/opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/hadoop-imageprocessor.jar:/opt/oracle/oracle-
spatial-graph/spatial/raster/jlib/hadoop-raster-fwk-
api.jar:/opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/gdal.jar:$HADOOP_CLASSPATH
```

Step 11: Right click on the project and select Project Properties. In the emergent window select Deployment option and click on Create Deployment Profile icon. Select JAR File as the Profile Type and set examples as the Deployment Profile Name, click OK to Finish.



Step 12: In the emergent Edit JAR Deployment Profile Properties, set the main class to oracle.raster.test.RasterExample and click Ok. Set the JAR File output path to **/opt/oracle/oracle-spatial-graph/spatial/raster/jlib**, the name to examples.jar and click Ok to accept all changes.

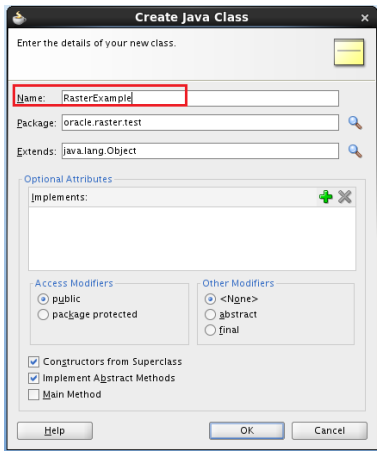


Load a set of rasters and process the mosaic operation.

In these steps you will load in HDFS a set of rasters with different resolutions and SRIDs in a first job, and in a second job you will process these rasters by creating a mosaic with them.

1. Go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster** and review the rasters included there. There are three Hawaii rasters in different resolutions; the three of them will be loaded in this exercise and have 3 bands. There is also an Elevation Model for Napa that is not planned to be loaded in this exercise.

2. Right click on the RasterTest project in the Application Navigator and select New Java Class. In the Create Java Class window, set *RasterExample* as the Name for the class and click Ok.



3. Let´s start programming the RasterExample.java class; Create a *main* method to be the input point of the class(public static void main(String args[])), all the code will set inside this method. Next create a HadoopConfiguration object and set the gdal data path as well as the jar file that contains the mapreduce code to load the rasters. Let the JDeveloper work on the import statements.

```
HadoopConfiguration hadoopConf = new HadoopConfiguration();
String gdalData = "/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data";
hadoopConf.setGdalDataPath(gdalData);
hadoopConf.setMapreduceJobJar("hadoop-imageloader.jar");
```

4. Now create a HadoopLoader object and set the rasters to load.

```
String rasterDirectory = "/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/hawaii.tif,/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/kahoolawe.tif,/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/maui.tif ";

RasterLoaderJob loader = (RasterLoaderJob) hadoopConf.createRasterLoaderJob();
loader.setFilesToLoad(rasterDirectory);
```

5. The rasters will be loaded in the HDFS folder you specify, for this exercise we are using ***exercise1*** folder. This directory will be located in the following path: /user/*hdfs*/exercise1/. As you can notice, hdfs directory is included in the path, since that is the user executing the loading process.

```
loader.setOutputFolder("exercise1");
```

6. You can also specify a thumbnail directory, where a thumbnail of every loaded raster is stored when the job finishes. You can use it to verify if the raster loaded correctly and no information was lost. For this exercise we are setting ***/opt/shareddir/spatial*** as the thumbnail folder, using the –thumbnail option when executing the job.

```
loader.setRasterThumbnailFolder("/opt/shareddir/spatial ");
```

7. Another optional setting is the number of overlapping pixels, which indicate the pixels from the adjacent tiles that will be shared in every tile, these pixels apply to all directions (bottom, top, left and right). For this exercise we are setting this value as ***50***.

```
loader.setTileOverlap("50");
```

8. Set Gdal directory to **/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib** and execute the job.

```
String gdalDirectory =    "/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib";

try {
    loader.setGdalPath(gdalDirectory);
    //Executes the job
    boolean loaderSuccess = loader.execute();

    if (loaderSuccess) {
            System.out.println("Successfully executed loader job");
    } else {
            System.out.println("Failed to execute loader job");
    }
  } catch (Exception e) {
 System.out.println("Problem when trying to execute raster loader " + e.getMessage());
}
```

9. The raster loader code is ready, and by this execution step, rasters should be tiled and loaded, the next step is to process a mosaic operation using them. Let´s start with the mosaic process, all the mosaic code should be inside the *if(loaderSuccess).* Assign the variable hadoopConf to a new HadoopConfiguration object, set gdalData path , same used for loader configuration object and set the jar name to hadoop-imageprocessor.jar.

```
hadoopConf = new HadoopConfiguration();
hadoopConf.setGdalDataPath(gdalData);
```

```
hadoopConf.setMapreduceJobJar("hadoop-imageprocessor.jar");
```

10. Now create a RasterProcessorJob and set the gdal directory, with the same value used for loader.

```
RasterProcessorJob processor = (RasterProcessorJob) hadoopConf.createRasterProcessorJob();
processor.setGdalPath(gdalDirectory);
```

11. Now create a MosaicConfiguration object and set the number of bands to 3, set the output directory to **/opt/shareddir/spatial**, and name of the file to ***APIMosaic***. Set the output filesystem to NFS, using the value ***RasterProcessorJob.FS***, raster output to ***GTIFF*** and SRID to ***26904***.

```
MosaicConfiguration mosaic = new MosaicConfiguration();
mosaic.setBands(3);
mosaic.setDirectory("/opt/shareddir/spatial");
mosaic.setFileName("APIMosaicFS");
mosaic.setFileSystem(RasterProcessorJob.FS);
mosaic.setFormat("GTIFF");
 mosaic.setSrid("26904");
```

12. Now let´s set the mosaic features in terms of resolution, data type and coordinated that will cover. The NODATA value will be used for pixels that do not intersect any of the raster sources. We will use the FILE_LENGTH algorithm to order the source rasters in the descending order.

```
mosaic.setWidth(1600);
mosaic.setHeight(986);
//byte datatype
mosaic.setPixelType("1");
mosaic.setNoData("#FFFFFF");
 //in case two or more rasters overlap, the area covered by the rasters will determine its priority in the mosaic
mosaic.setOrderAlgorithm(ProcessConstants.ALGORITHM_FILE_LENGTH);
//rasters that cover less area will be located on top of the mosaic, DESC, descending order
mosaic.setOrder(RasterProcessorJob.DESC);

 //width for pixels in X and Y
mosaic.setPixelXWidth(280.388143);
mosaic.setPixelYWidth(-280.388143);
//upper left coordinates
mosaic.setUpperLeftX(556958.985610);
mosaic.setUpperLeftY(2350324.082505);
```

13. Now that you specified the features of the mosaic you must create a catalog with the rasters that will be considered for mosaic creation. Below is the code to create the first raster object, and is your job to create the other two objects for maui and kahoolawe. The Raster class package is oracle.spatial.hadoop.rasterapi.core.beans

```
RasterCatalog catalog = new RasterCatalog();

//Creates a raster object for the catalog
Raster raster = new Raster();
raster.setBands(3);
//the tree bands will appear in order 1,2, 3. You may list less bands here.
raster.setBandsOrder("1,2,3");
raster.setDataType(1);
raster.setRasterLocation("/user/hdfs/exercise1/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/hawaii.tif.ohif");
```

```
//Add raster to catalog
catalog.addRasterToCatalog(raster);

//Create and add the other two rasters for maui and kahoolawe
```

14. MosaicConfiguration and RasterCatalog objects are ready, the last step is to set them to the processor job.

```
processor.setMosaicConfigurationObject(mosaic.getCompactMosaic());
processor.setCatalogObject(catalog.getCompactCatalog());
```

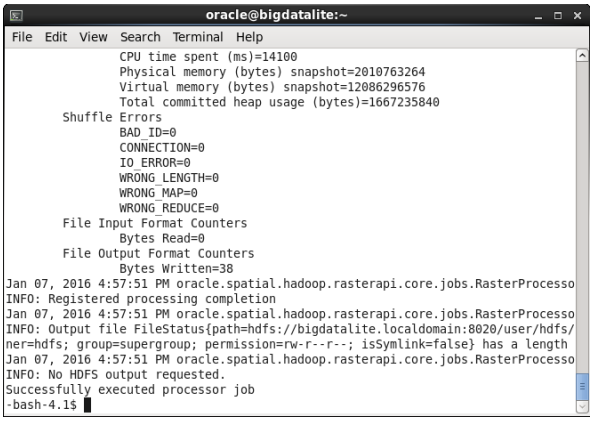
15. Now that the catalog and mosaic configuration are set, start the job using the execute command provided by RasterProcessorJob.

```
boolean processorSuccess = processor.execute();
if (processorSuccess) {
   System.out.println("Successfully executed processor job");
} else {
   System.out.println("Failed to execute processor job");
}
```

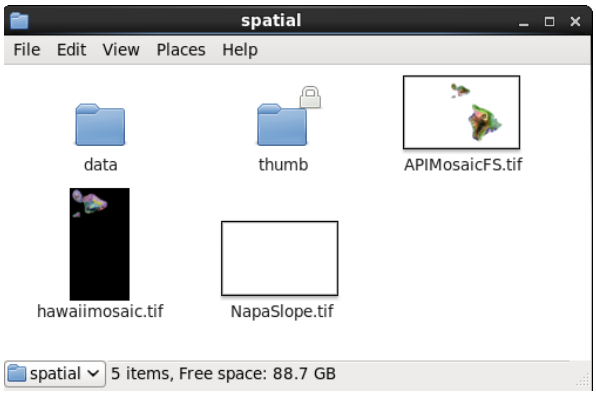
16. The code is ready, now let´s build it and pack a jar file to execute it using <Ctrl> + <F9>, after it is built, right click on the project select Deploy and the examples profile you just created. Jar file is deployed and ready to use.

17. To execute the process go to the command line window where you switched to hdfs user and execute the class you created and wait until both jobs complete.

```
hadoop jar /opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/examples.jar
```



18. Review the mosaic output by listing the directory you set as mosaic output path in the example class, **/opt/shareddir/spatial**. As you can see, APIMosaicFS.tif includes the three rasters you added in the processor job.

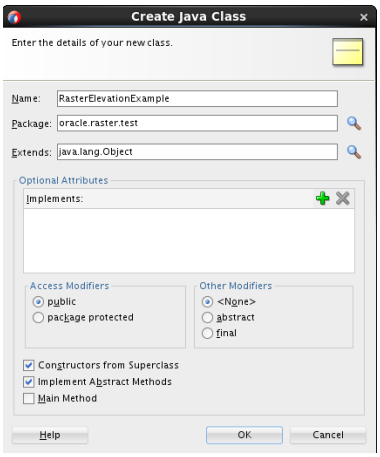


Load a DEM and process slope function.

In these steps you will load in HDFS a DEM in a first job, and in a second job you will process these rasters by creating a mosaic with them.

1. Go to **/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster** and review the rasters included there. NapaDEM.tif is the raster we will load in this example, it is an Elevation Model of Northern Napa Valley in SRID 32610 and has a single band.

2. Right click on the RasterTest project in the Application Navigator and select New Java Class. In the Create Java Class window, set *RasterElevationExample* as the Name for the class and click Ok.



3. Let´s start programming the RasterElevationExample.java class; Create a *main* method to be the input point of the class(public static void main(String args[])), all the code will be set inside this method. Next create a HadoopConfiguration object and set the gdal data path as well as the jar file that contains the mapreduce code to load the rasters. Let JDeveloper work on the import statements.

```
HadoopConfiguration hadoopConf = new HadoopConfiguration();
String gdalData = "/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/data";
hadoopConf.setGdalDataPath(gdalData);
hadoopConf.setMapreduceJobJar("hadoop-imageloader.jar");
```

4. Now create a HadoopLoader object and set the rasters to load.

```
String rasterDirectory = "/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/NapaDEM.tif ";

RasterLoaderJob loader = (RasterLoaderJob) hadoopConf.createRasterLoaderJob();
loader.setFilesToLoad(rasterDirectory);
```

5. The raster will be loaded in the HDFS folder you specify, for this exercise we are using ***exercise2*** folder. This directory will be located in the following path: /user/*hdfs*/exercise2/. As you can notice, hdfs directory is included in the path, since that is the user executing the loading process.

```
loader.setOutputFolder("exercise2");
```

6. You can also specify a thumbnail directory, where a thumbnail of every loaded raster is stored when the job finishes. You can use it to verify if the raster loaded correctly and no information was lost. For this exercise we are setting ***/opt/shareddir/spatial*** as the thumbnail folder, using the –thumbnail option when executing the job.

```
loader.setRasterThumbnailFolder( "/opt/shareddir/spatial");
```

7. Another optional setting is the number of overlapping pixels, which indicate the pixels from the adjacent tiles that will be shared in every tile, these pixels apply to all directions (bottom, top, left and right). For this exercise we are setting this value as ***30***.

```
loader.setTileOverlap("30");
```

8. Set Gdal directory to ***/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib*** and execute the job.

```
String gdalDirectory =    "/opt/oracle/oracle-spatial-graph/spatial/raster/gdal/lib";
try {
    loader.setGdalPath(gdalDirectory);
    //Executes the job
    boolean loaderSuccess = loader.execute();

    if (loaderSuccess) {
                System.out.println("Successfully executed loader job");
    } else {
                System.out.println("Failed to execute loader job");
    }
  } catch (Exception e) {
  System.out.println("Problem when trying to execute raster loader " + e.getMessage());
}
```

9. The raster loader code is ready, and by this execution step, elevation model should be tiled and loaded, the next step is to process a mosaic operation using it and calculate the slope for every pixel. Let´s start with the mosaic process, all the mosaic code should be inside the *if(loaderSuccess).* Assign the variable hadoopConf to a new HadoopConfiguration object, set gdalData path , same used for loader configuration object and set the jar name to hadoop-imageprocessor.jar.

```
hadoopConf = new HadoopConfiguration();
hadoopConf.setGdalDataPath(gdalData);
hadoopConf.setMapreduceJobJar("hadoop-imageprocessor.jar");
```

10. Now create a RasterProcessorJob and set the gdal directory, with the same value used for loader.

```
RasterProcessorJob processor = (RasterProcessorJob) hadoopConf.createRasterProcessorJob();
processor.setGdalPath(gdalDirectory);
```

11. Now create a MosaicConfiguration object and set the number of bands to 3, set the output directory to ***/opt/shareddir/spatial***, and name of the file to ***APIMosaicSlope***. Set the output filesystem to NFS, using the value RasterProcessorJob.FS, raster output to ***GTIFF*** and SRID to ***32610***.

```
MosaicConfiguration mosaic = new MosaicConfiguration();
mosaic.setBands(1);
mosaic.setDirectory("/opt/shareddir/spatial");
mosaic.setFileName("APIMosaicSlope");
mosaic.setFileSystem(RasterProcessorJob.FS);
mosaic.setFormat("GTIFF");
mosaic.setSrid("32610");
```

12. Now let´s set the mosaic features in terms of resolution, data type and coordinated that will cover. The NODATA value will be used for pixels that do not intersect any of the raster sources. We will use the FILE_LENGTH algorithm to order the source rasters in the descending order.

```
mosaic.setWidth(1400);
mosaic.setHeight(870);
//float32 datatype
mosaic.setPixelType("6");
mosaic.setNoData("#00");
 //in case two or more rasters overlap, the area covered by the rasters will determine its priority in the mosaic
mosaic.setOrderAlgorithm(ProcessConstants.ALGORITHM_FILE_LENGTH);
//rasters that cover less area will be located on top of the mosaic, DESC, descending order
mosaic.setOrder(RasterProcessorJob.DESC);

 //width for pixels in X and Y
mosaic.setPixelXWidth(48.416923);
mosaic.setPixelYWidth(-48.416923);
//upper left coordinates
mosaic.setUpperLeftX(520284.694956);
mosaic.setUpperLeftY(4305151.843027);
```

13. You also need to specify the processing class that will calculate the slope for every pixel.

```
 //Sets slope processing class to mosaic configuration
mosaic.setProcessingClasses("oracle.spatial.hadoop.imageprocessor.process.ImageSlope");
```

14. Now that you specified the features of the mosaic you must create a catalog with the rasters that will be considered for mosaic creation. Below is the code to create the raster object and add it to the catalog. The Raster class package is oracle.spatial.hadoop.rasterapi.core.beans

```
 RasterCatalog catalog = new RasterCatalog();

 //Creates a raster object for the catalog
 Raster raster = new Raster();
 raster.setBands(1);
 raster.setBandsOrder("1");
 raster.setDataType(6);
raster.setRasterLocation("/user/hdfs/exercise2/opt/oracle/oracle-spatial-graph/spatial/raster/Raster-HOL/data/raster/NapaDEM.tif.ohif");
 //Add raster to catalog
 catalog.addRasterToCatalog(raster);
```

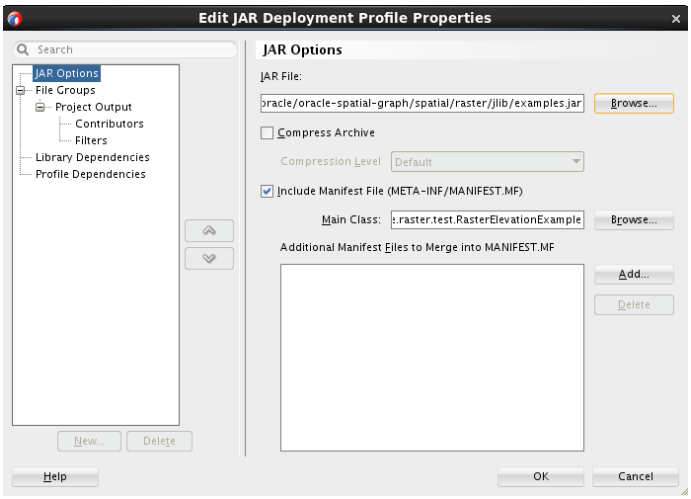
15. MosaicConfiguration and RasterCatalog objects are ready, the last step is to set them to the processor job.

```
 processor.setMosaicConfigurationObject(mosaic.getCompactMosaic());
 processor.setCatalogObject(catalog.getCompactCatalog());
```

16. Now that the catalog and mosaic configuration are set, start the job using the execute command provided by RasterProcessorJob.

```
 boolean processorSuccess = processor.execute();
if (processorSuccess) {
   System.out.println("Successfully executed processor job");
} else {
   System.out.println("Failed to execute processor job");
}
```

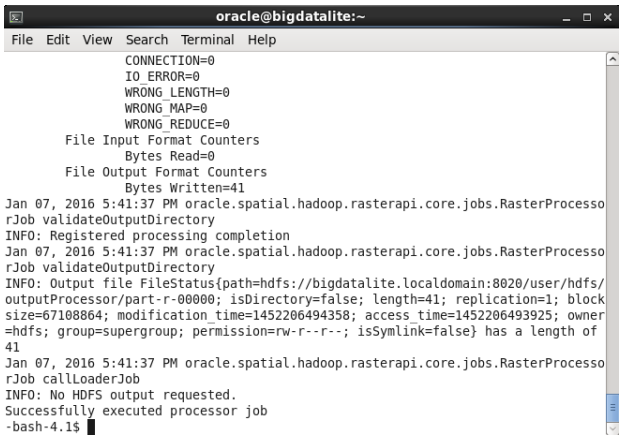
17. The code is ready, now let´s build it and pack a jar file to execute it. Right click on the project and select Project Properties. In the emergent window select Deployment option and click on the examples deployment profile, click the Edit Profile icon  . In the emergent Edit JAR Deployment Profile Properties, click the Browse button to set the main class to oracle.raster.test.RasterElevationExample and click Ok to accept all changes.



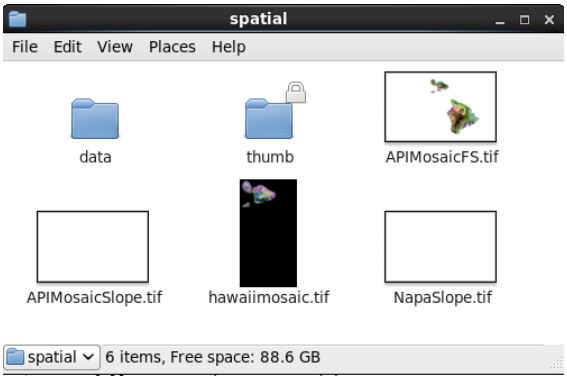
18. Deployment profile is ready, build the project using <Ctrl> + <F9>, after it is built, right click on the project select Deploy -> examples to JAR file. Jar file is deployed and ready to use.

19. To execute the process go to the command line window where you switched to hdfs user, execute the class you created and wait until both jobs complete. $HADOOP_CLASSPATH environment variable was already modified in Preparation Steps at the beginning of this Lab Part to include all the jars required for job execution.

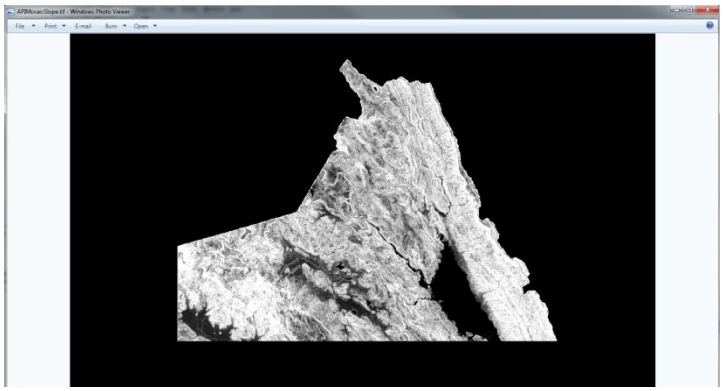
```
hadoop jar /opt/oracle/oracle-spatial-
graph/spatial/raster/jlib/examples.jar
```



20. Review the mosaic output by listing the directory you set as mosaic output path in the example class, /opt/shareddir/spatial.



21. The output raster with the slope cannot be visualized in Linux environment for the standard image tools, if you want to do so you may want to open the raster in Windows environment with Windows Photo Viewer. If you don't have access to Windows environment you may try any specialized raster tool.



Concluding comments

Thank you for participating in this hands-on lab. More examples are available under the folders `/opt/oracle/oracle-spatial-graph/spatial/vector/examples` and `/opt/oracle/oracle-spatial-graph/spatial/raster/examples`. More information can also be found in the following page:

http://www.oracle.com/technetwork/database/database-technologies/bigdata-spatialandgraph/overview/index.html