**Building Custom Drill Function using Drill API in Java**

**Prerequisites :**

* [Oracle Java SE Development (JDK) Kit 7](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html) or later
* [Apache Drill 1.1](http://getdrill.org/drill/download/apache-drill-1.1.0.tar.gz) or later
* [Maven 3.0](https://maven.apache.org/download.cgi) or later

Two types of Functions can be build – Simple or Aggregate Functions

Simple Function operates on single row and produces single row as output.

Aggregate Function operates on multiple input rows and produces single row as output. Groupby clause is needed along with Aggregate Function to produce the output.

**Step : 1**

**Adding Dependencies in Maven**

<dependency>

<groupId>org.apache.drill.exec</groupId>

<artifactId>drill-java-exec</artifactId>

<version>1.1.0</version>

</dependency>

**Step : 2**

**Adding Annotations to the function template**

@FunctionTemplate(

name = "randomco",

scope = FunctionTemplate.FunctionScope.*SIMPLE*,

nulls = FunctionTemplate.NullHandling.*NULL\_IF\_NULL*

)

Here,name will be the Function name to be used in Drill query, scope specifies whether it is Simple or Aggregate and nulls to decide how to handle the null values.

Eg : Select RANDOMCO(…) from Testtable;

**Step : 3**

**Writing Java Class**

Write a java class which implements DrillSimpleFunc. This interface has two methods which needs to be overridden. Setup() and eval().

(In this case, I am writing a Custom function which will derive a random latitude and longitude values from set of geo lat,long columns. )

Define Input Params and Output Params which will be passed to this function.

In this case there are 8 columns to be taken as input parameters.

If Parameter passing to function is a constant instead of Column name . We need to specify for example : @Param(constant = true).

Because the output is varchar , we need to inject a buffer that drill uses for output.

**public** **class** RandomCoordinates **implements** DrillSimpleFunc {

@Param

VarCharHolder lat\_1;

@Param

VarCharHolder long\_1;

@Param

VarCharHolder lat\_2;

@Param

VarCharHolder long\_2;

@Param

VarCharHolder lat\_3;

@Param

VarCharHolder long\_3;

@Param

VarCharHolder lat\_4;

@Param

VarCharHolder long\_4;

@Output

VarCharHolder out;

@Inject

DrillBuf buffer;

**public** **void** setup() {}

Public void eval(){}

}

Step 4:

Building the required logic inside eval method.

**public** **void** eval() {

//Get 8 coordinates from input and derive a random coordinate

**float** rlat = (**float**) 0.0;

**float** rlong = (**float**) 0.0;

**float** lat1 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(lat\_1));

**float** long1 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(long\_1));

**float** lat2 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(lat\_2));

**float** long2 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(long\_2));

**float** lat3 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(lat\_3));

**float** long3 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(long\_3));

**float** lat4 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(lat\_4));

**float** long4 = Float.*parseFloat*(org.apache.drill.exec.expr.fn.impl.StringFunctionHelpers.*getStringFromVarCharHolder*(long\_4));

// Logic to build Random Coordinates from geo boundary coordinates

**if**(lat4 > lat2)

{

rlat = (**float**) (Math.*random*() \* (lat4 - lat2) + lat2);

}

**else**

{

rlat = (**float**) (Math.*random*() \* (lat2 - lat4) + lat4);

}

**if**( long3 > long1)

{

rlong = (**float**) Math.*random*() \* (long3 - long1) + long1;

}

**else**

{

rlong = (**float**) Math.*random*() \* (long1 - long3) + long3;

}

String outputvalue = rlat + "," + rlong;

// put the output value in the out buffer

out.buffer = buffer;

out.start = 0;

out.end = outputvalue.getBytes().length;

buffer.setBytes(0, outputvalue.getBytes());

}

**Complete Code :**

<https://github.com/n00bc0der89/DrillCustomFunction.git>

**Preparing Package :**

For drill, both source and class are required for necessary code generation. Drill uses the compiled code to access the annotations and uses the source code to do code generation.

In pom.xml

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-source-plugin</artifactId>

<version>2.4</version>

<executions>

<execution>

<id>attach-sources</id>

<phase>package</phase>

<goals>

<goal>jar-no-fork</goal>

</goals>

</execution>

</executions>

</plugin>

Add a drill-module.conf File to Resources

Add a drill-module.conf file in the resources folder of your project. The presence of this file tells Drill that your jar contains a custom function. Put the following line in the drill-module.config:

drill {

classpath.scanning {

base.classes : ${?drill.classpath.scanning.base.classes} [

org.customdrill.drill\_simple\_randomco.RandomCoordinates

],

packages : ${?drill.classpath.scanning.packages} [

org.customdrill.drill\_simple\_randomco

]

}

}

Build and Deploy the Function

Build the function using mvn package:

mvn clean package

Maven generates two JAR files:

The default jar with the classes and resources (drill-simple-randomco-1.0.jar)

A second jar with the sources (drill-simple-randomco-1.0-sources.jar)

Add the JAR files to Drill, by copying them to the following location:

**<Drill installation directory>/jars/3rdparty**

Once adding the jar files, Restart drill and test the query.

Select RANDOMCO(

place\_coordinate1\_lat,

place\_coordinate1\_lng,

place\_coordinate2\_lat,

place\_coordinate2\_lng,

place\_coordinate3\_lat,

place\_coordinate3\_lng,

place\_coordinate4\_lat,

place\_coordinate4\_lng) as Random,

place\_coordinate1\_lat,

place\_coordinate1\_lng,

place\_coordinate2\_lat,

place\_coordinate2\_lng,

place\_coordinate3\_lat,

place\_coordinate3\_lng,

place\_coordinate4\_lat,

place\_coordinate4\_lng from `hive\_social\_media`.`default`.`newtwittercategorystream` where place\_coordinate1\_lat <> 'null' limit 3

Output :



**Dynamic UDFS**

In case the Custom Function jars which was created above are manually added into jar folders of Drill Installed path. One must have to restart the drillbit so that custom function gets added in classpath of drill and can be used in drill query. However, it disrupt users, when administrators manually load and unload UDFs in a multi-tenant environment.

To avoid it, we can configure the drill to have a staging area where we can add our custom jars instead of adding into build-in jars directory of drill.

Adding our jars into staging area and registering the jars from command –

CREATE FUNCTION USING JAR ‘<jar\_name.jar>’

Allows us to use the function in our query without restarting drillbit.

Steps:

Dynamic UDF feature is enabled by default. An administrator can enable or disable the feature using the ALTER SYSTEM SET command with the exec.udf.enable\_dynamic\_support option.

Run this command in drill

ALTER SYSTEM SET `**exec**.**udf**.**enable\_dynamic\_support**` = true

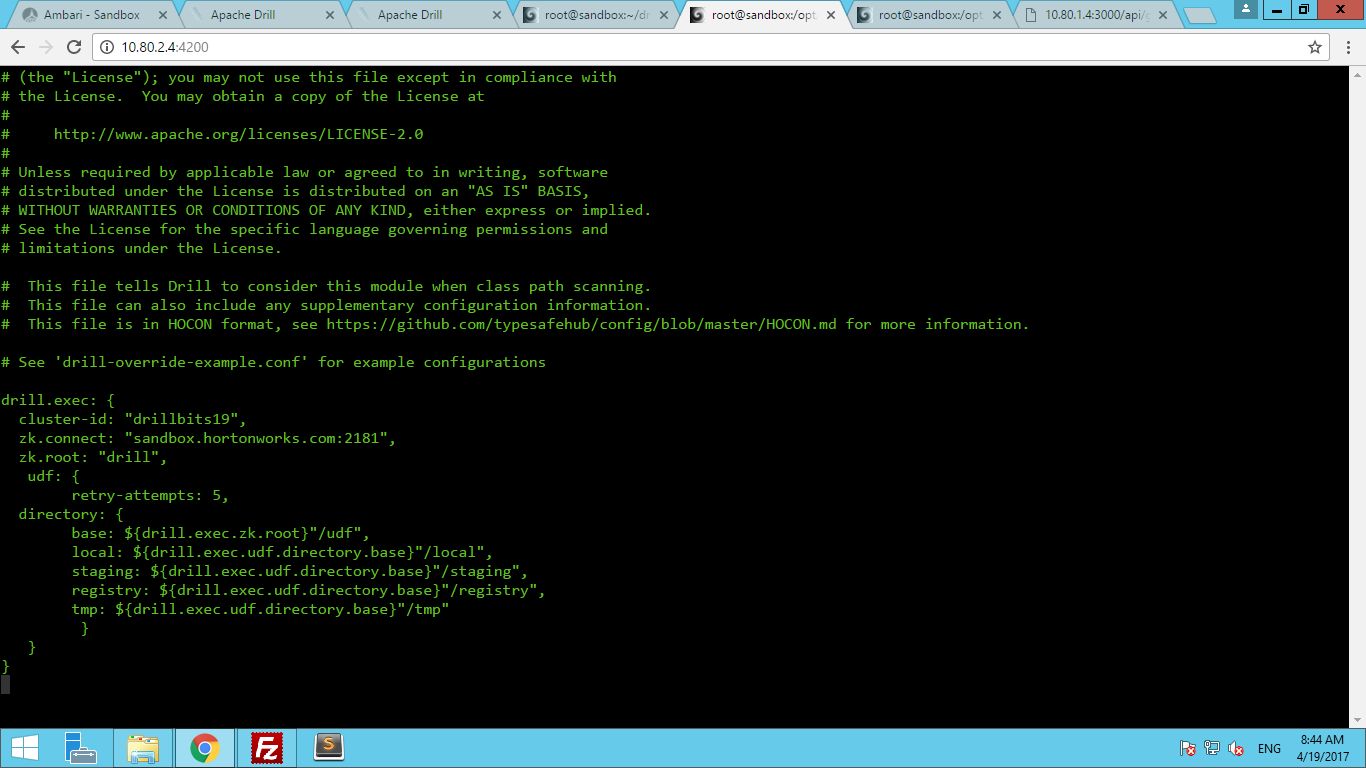
1. In config folder of Apache drill, Edit drill-override.conf file with below properties.

drill.exec:

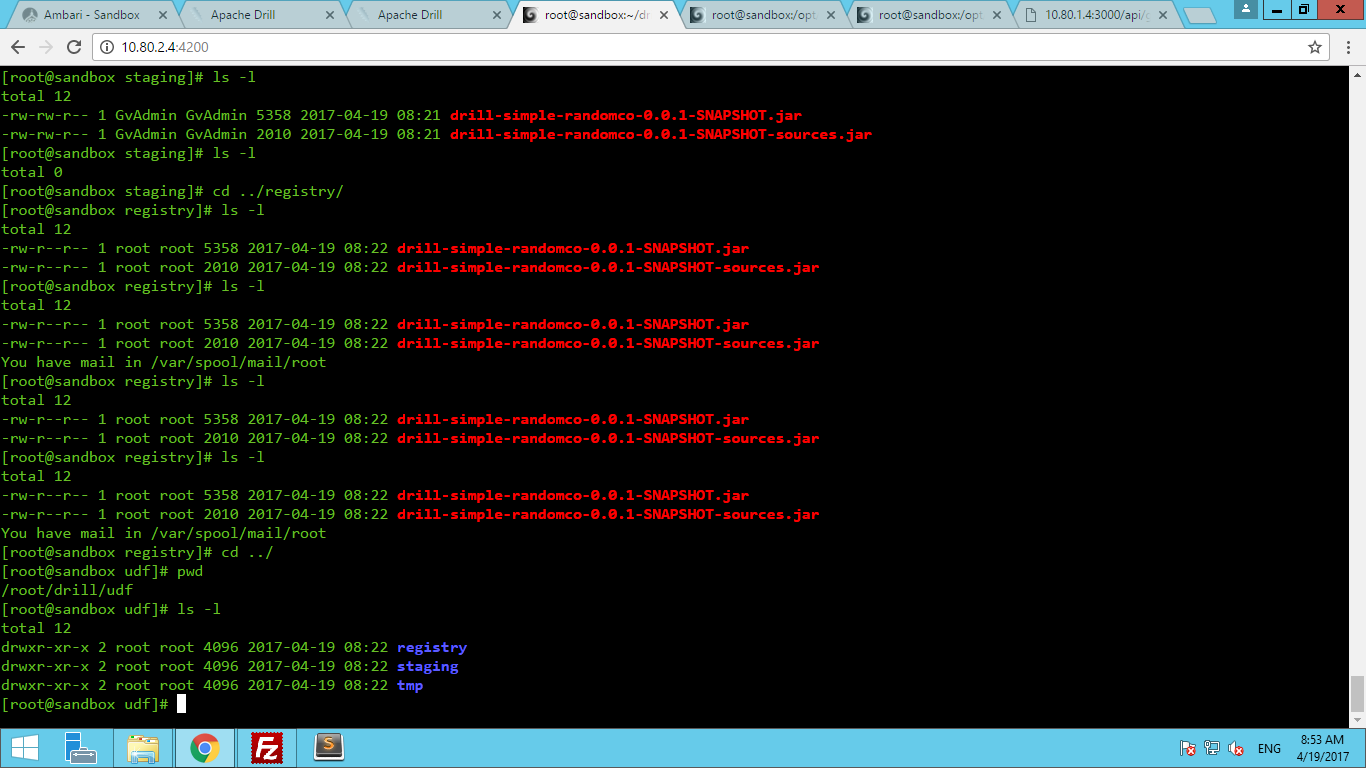
{ cluster-id:"drillbits19", zk.connect:"sandbox.hortonworks.com:2181", zk.root:"drill", udf: { retry-attempts:5, directory: { base: ${drill.exec.zk.root}"/udf", local: ${drill.exec.udf.directory.base}"/local", staging: ${drill.exec.udf.directory.base}"/staging", registry: ${drill.exec.udf.directory.base}"/registry", tmp: ${drill.exec.udf.directory.base}"/tmp" }

}

}



Restart the drillbit once after editing above file, so that it creates the directories in /root path of Local system if directories are not created by default.



So /root/drill folder will have directory structures as mentioned in config files.

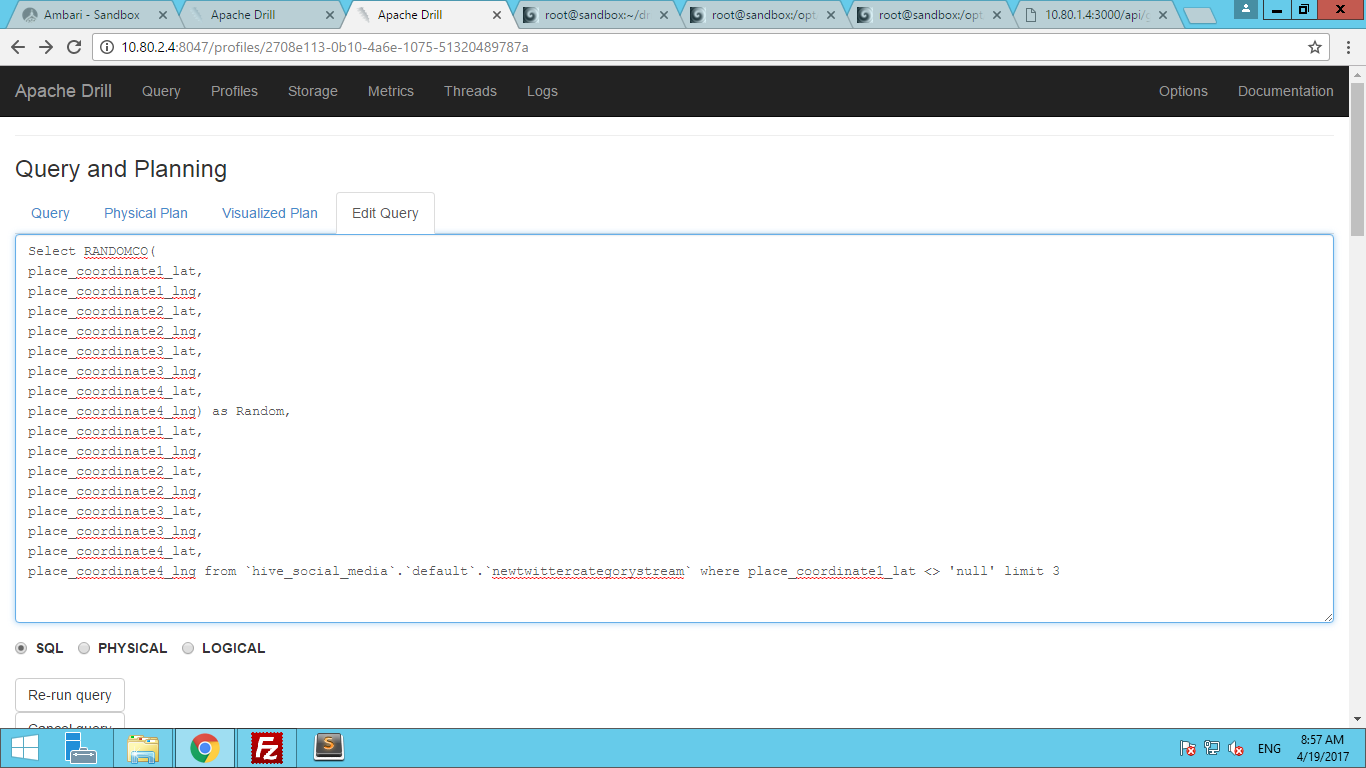
Now, copy the jar files(both source and class) into staging directory.

Now Run the command,

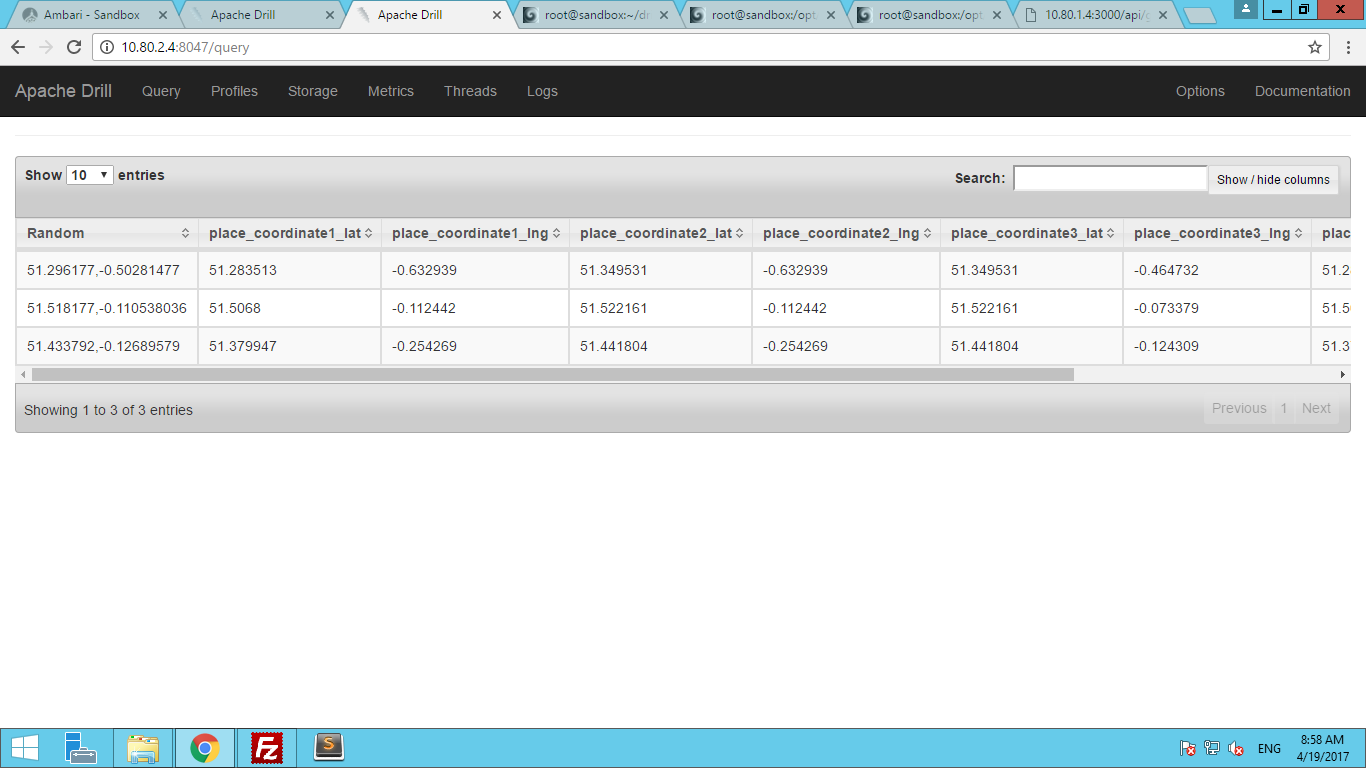
CREATE FUNCTION USING JAR ‘drill-simple-randomco-0.0.1-SNAPSHOT.jar’

This will move the jars from staging area into registry directory.

Now run the query in drill using this custom Function



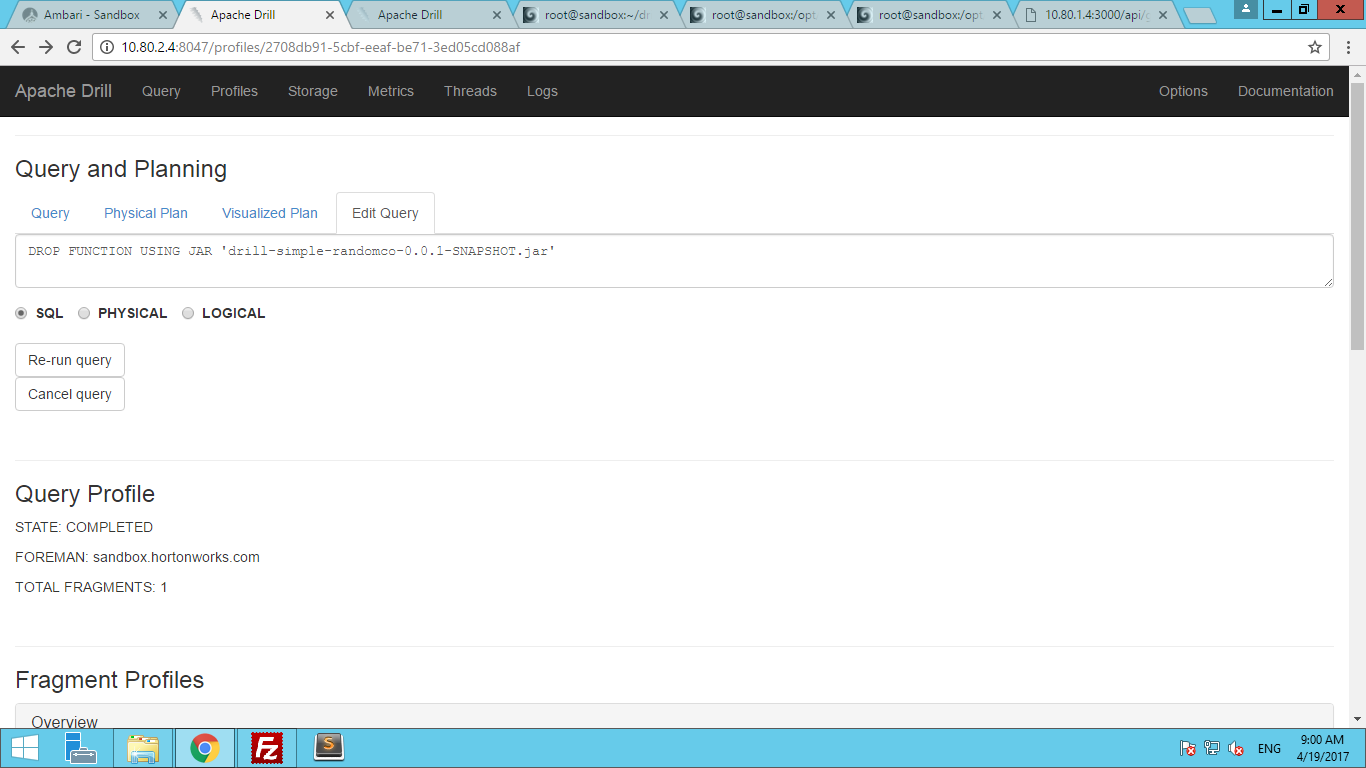
Output :

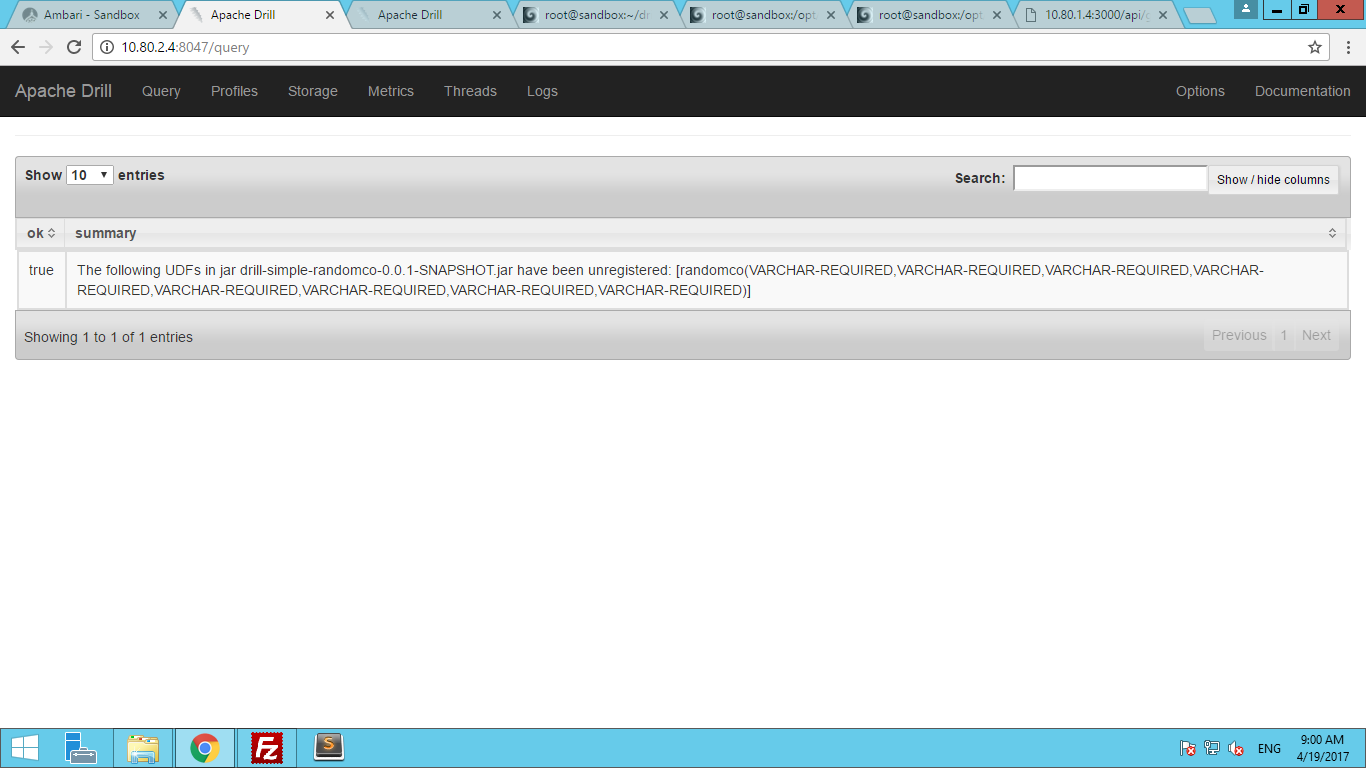


To Unregister the jars. Run Command

DROP FUNCTION USING JAR ‘drill-simple-randomco-0.0.1-SNAPSHOT.jar’

This command removes the jars from registry directory. And firing query using custom function will result in error.





Query Output :

