**Lab 3**

**Real Time data Ingestion in Hbase and Hive using Storm**

**Introduction**

In this tutorial, we will build a solution to ingest real time streaming data into HBase and HDFS.

In previous tutorial we have explored generating and processing streaming data with [Apache Kafka](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/) and [Apache Storm](http://hortonworks.com/hadoop-tutorial/ingesting-processing-real-time-events-apache-storm/). In this tutorial we will create HDFS Bolt & HBase Bolt to read the streaming data from the Kafka Spout and persist in Hive & HBase tables.

**Pre-Requisites**

* [Tutorial #1](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/) Simulate and Transport Real Time Events with Apache Kafka
* [Tutorial #2](http://hortonworks.com/hadoop-tutorial/ingesting-processing-real-time-events-apache-storm/) Ingest and Process Real Time Events with Apache Storm
* Downloaded and Installed the latest [Hortonworks Sandbox](http://hortonworks.com/products/hortonworks-sandbox/#install)
* [Learning the Ropes of the Hortonworks Sandbox](http://hortonworks.com/hadoop-tutorial/learning-the-ropes-of-the-hortonworks-sandbox/)

**Outline**

* [Hbase](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#hbase-concept-lab3)
* [Apache Storm](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#apache-storm-concept-lab3)
* [Step 1: Start Hbase](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#step-1-start-hbase-lab3)
* [Step 2: Create tables in HDFS and Hbase](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#step2-create-tables-hdfs-hbase-lab3)
* [Step 3: Launch new Storm topology](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#step3-launch-new-storm-topology-lab3)
* [Step 4: Generate events and verify data in HDFS and HBase](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#step4-generate-events-verify-data-hdfs-hbase-lab3)
* [Step 5: Code description](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#step5-code-description-lab3)
* [Appendix A: Update Tutorials Master Project](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#update-tutorials-master-project-lab3)
* [Appendix B: Enable Remote Desktop and Set up Storm Topology as an Eclipse Project](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#enable-remote-desktop-setup-topology-lab3)
* [Further Reading](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#further-reading-lab3)

**HBase**

HBase provides near real-time, random read and write access to tables (or to be more accurate 'maps') storing billions of rows and millions of columns.

In this case, once we store this rapidly and continuously growing dataset from Internet of Things (IoT), we will be able to perform a swift lookup for analytics regardless of the data size.

**Apache Storm**

Apache Storm is an Open Source distributed, reliable, fault–tolerant system for real time processing of large volume of data. Spout and Bolt are the two main components in Storm, which work together to process streams of data.

* **Spout**: Works on the source of data streams. In the "Truck Events" use case, Spout will read data from Kafka topics.
* **Bolt**: Spout passes streams of data to Bolt which processes and persists  it to a data store or sends it downstream to another Bolt.

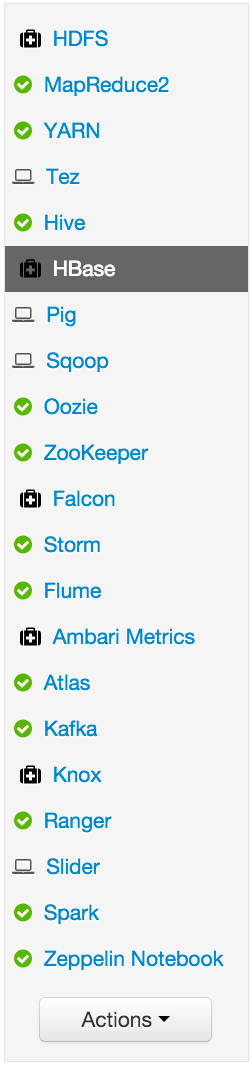
In this tutorial, you will learn the following topics:

* Configure Storm Bolt.
* Store Persisting data in HBase and Hive.
* Verify the data in HDFS and HBase.

**Step 1: Start HBase**

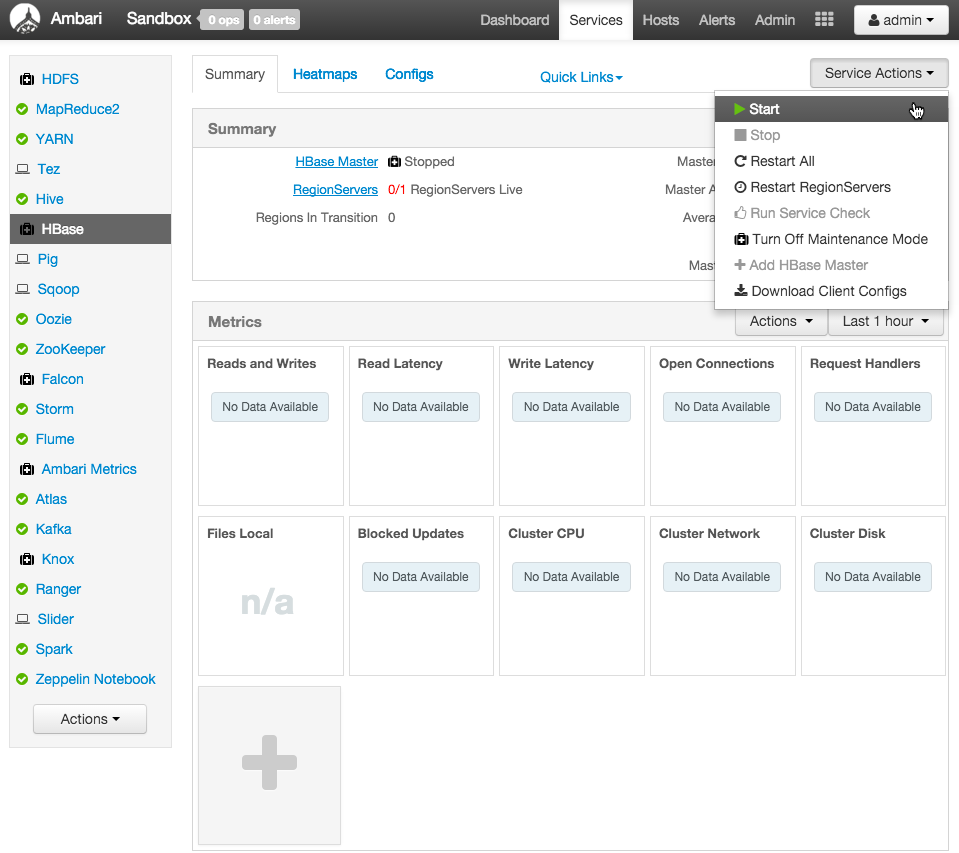
1. **View the HBase Services page**

Started by logging into Ambari as an admin user. From the previous tutorials: HDFS, Hive, YARN, Kafka and Storm should already be running but HBase may be down. From the Dashboard page of Ambari, click on HBase from the list of installed services.

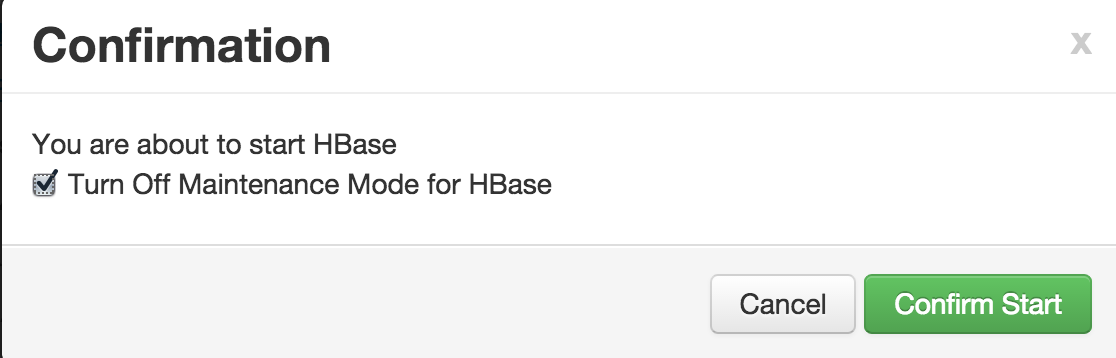


2. Start HBase

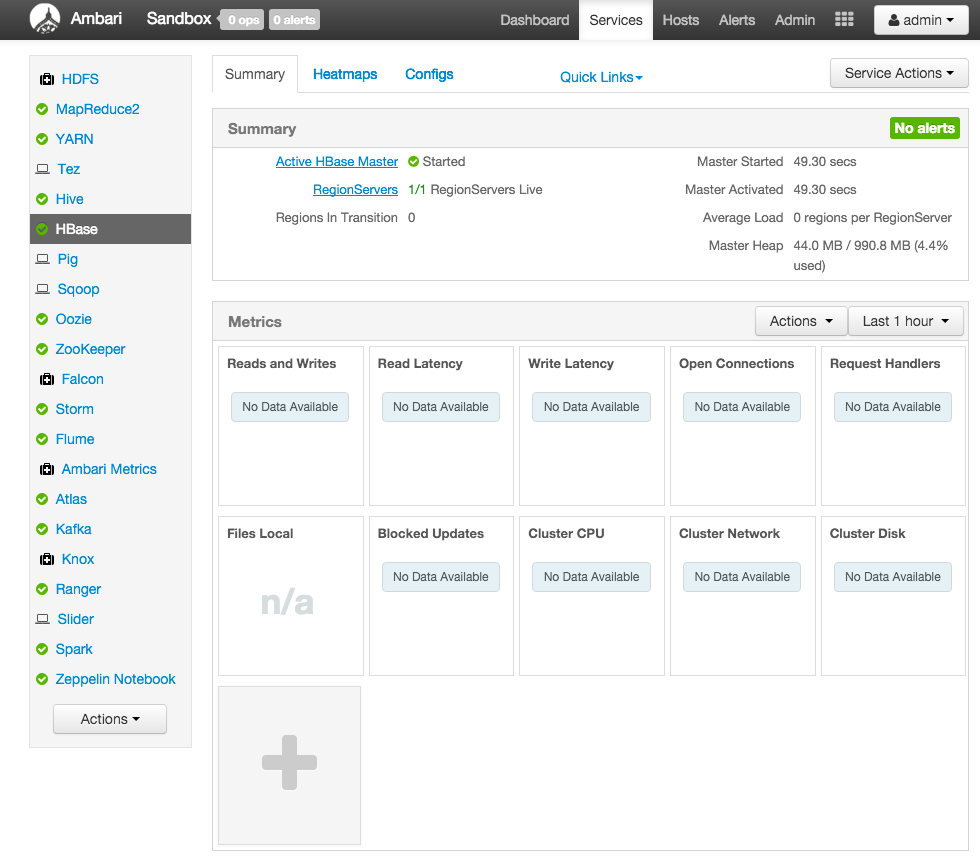
From the HBase page, click on Service Actions -> Start



Check the box and click on Confirm Start:



Wait for HBase to start (It may take a few minutes to turn green)



You can use the Ambari dashboard to check status of other components too. If **HDFS, Hive, YARN, Kafka, Storm or HBase** are down, you can start them in the same way: by selecting the service and then using the Service Actions to start it. The remaining components do not have to be up. (Oozie can be stopped to save memory, as it is not needed for this tutorial)

**Step 2: Create tables in HDFS & HBase**

* Create HBase tables

We will be working with 2 Hbase tables in this tutorial.

The first table stores all events generated and the second stores the 'driverId' and non-normal events count.

[root@sandbox ~]$ su hbase

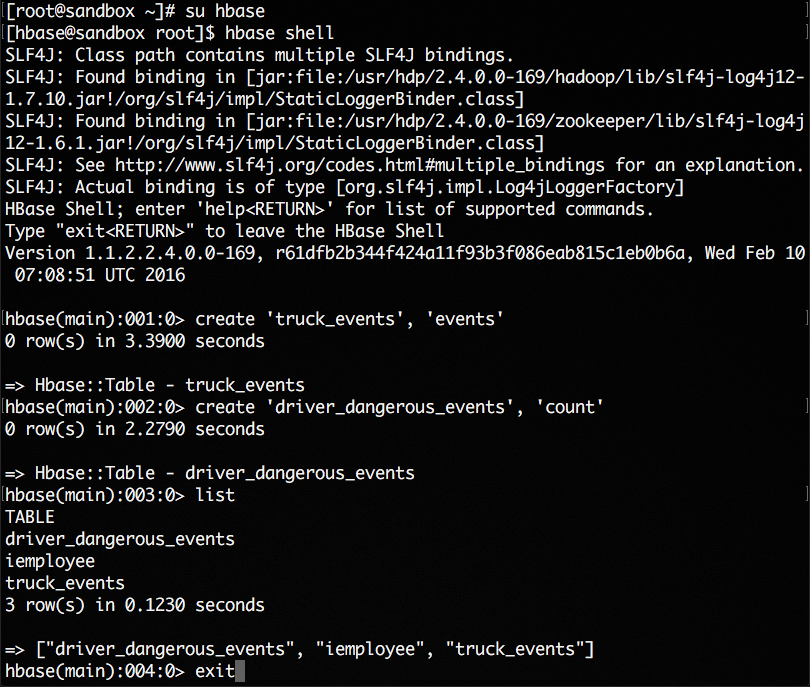
[hbase@sandbox root]$ hbase shell

hbase(main):001:0> create 'truck\_events', 'events'

hbase(main):002:0> create 'driver\_dangerous\_events', 'count'

hbase(main):003:0> list

hbase(main):004:0> exit



Next, we will create Hive tables.

* Create Hive tables

Open the Hive view in Ambari in a browser, copy the below script into the query editor and click Execute: [http://localhost:8080/#/main/views/HIVE/1.0.0/Hive](http://localhost:8080/#/main/views/HIVE/1.0.0/AUTO_HIVE_INSTANCE)

create table truck\_events\_text\_partition

(driverId string,

truckId string,

eventTime timestamp,

eventType string,

longitude double,

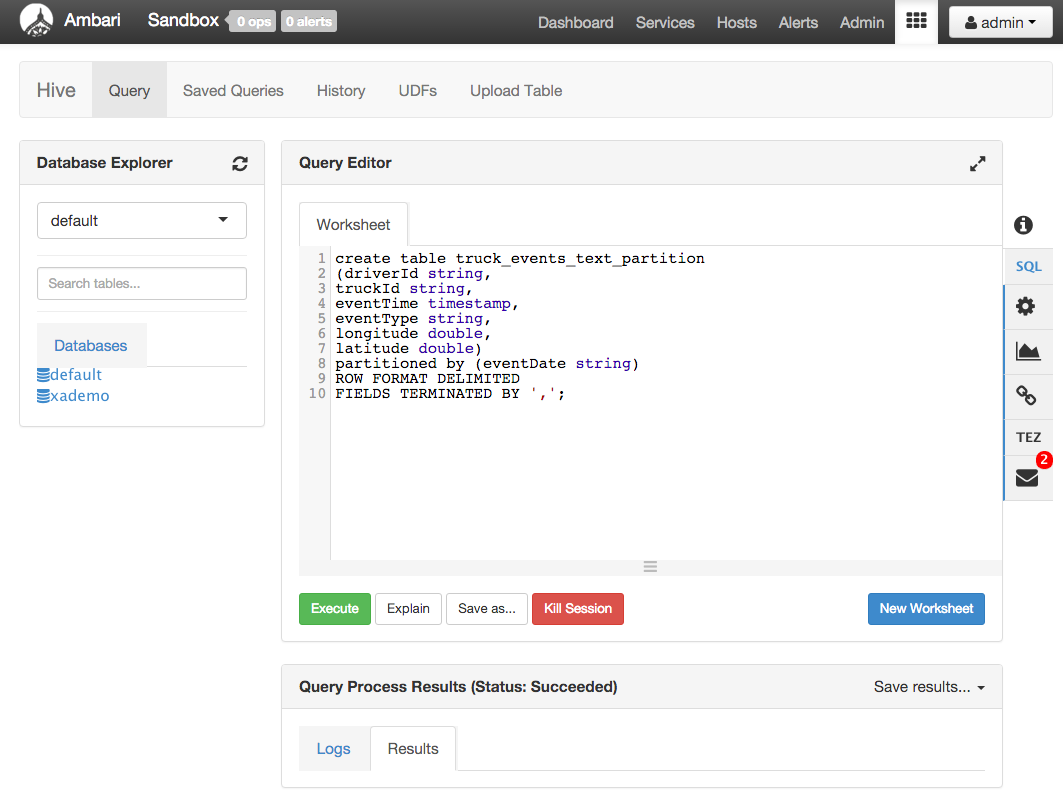
latitude double)

partitioned by (eventDate string)

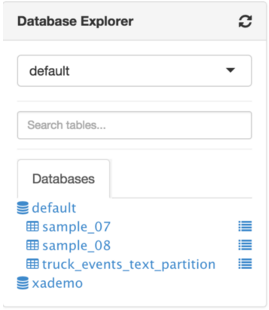
ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

This hive query creates the Hive table to persist all events generated. The table is partitioned by date.



Verify that the table has been properly created by refreshing the Database Explorer. Under Databases, click default to expand this table and the new table should appear. Clicking on the List icon next to truck\_events\_text\_partition shows that the table was created but empty.



* Create ORC 'truckevent' Hive tables

The Optimized Row Columnar (ORC) file format provides a highly efficient way to store Hive data. It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data.

Syntax for ORC tables:

CREATE TABLE … STORED AS ORC

ALTER TABLE … [PARTITION partition\_spec] SET FILEFORMAT ORC

**Note**: This statement only works on partitioned tables. If you apply it to flat tables, it may cause query errors.

Next let's create the 'truckevent' table as per the above syntax. Paste the below into the worksheet of the Hive view and click Execute

create table truck\_events\_text\_partition\_orc

(driverId string,

truckId string,

eventTime timestamp,

eventType string,

longitude double,

latitude double)

partitioned by (eventDate string)

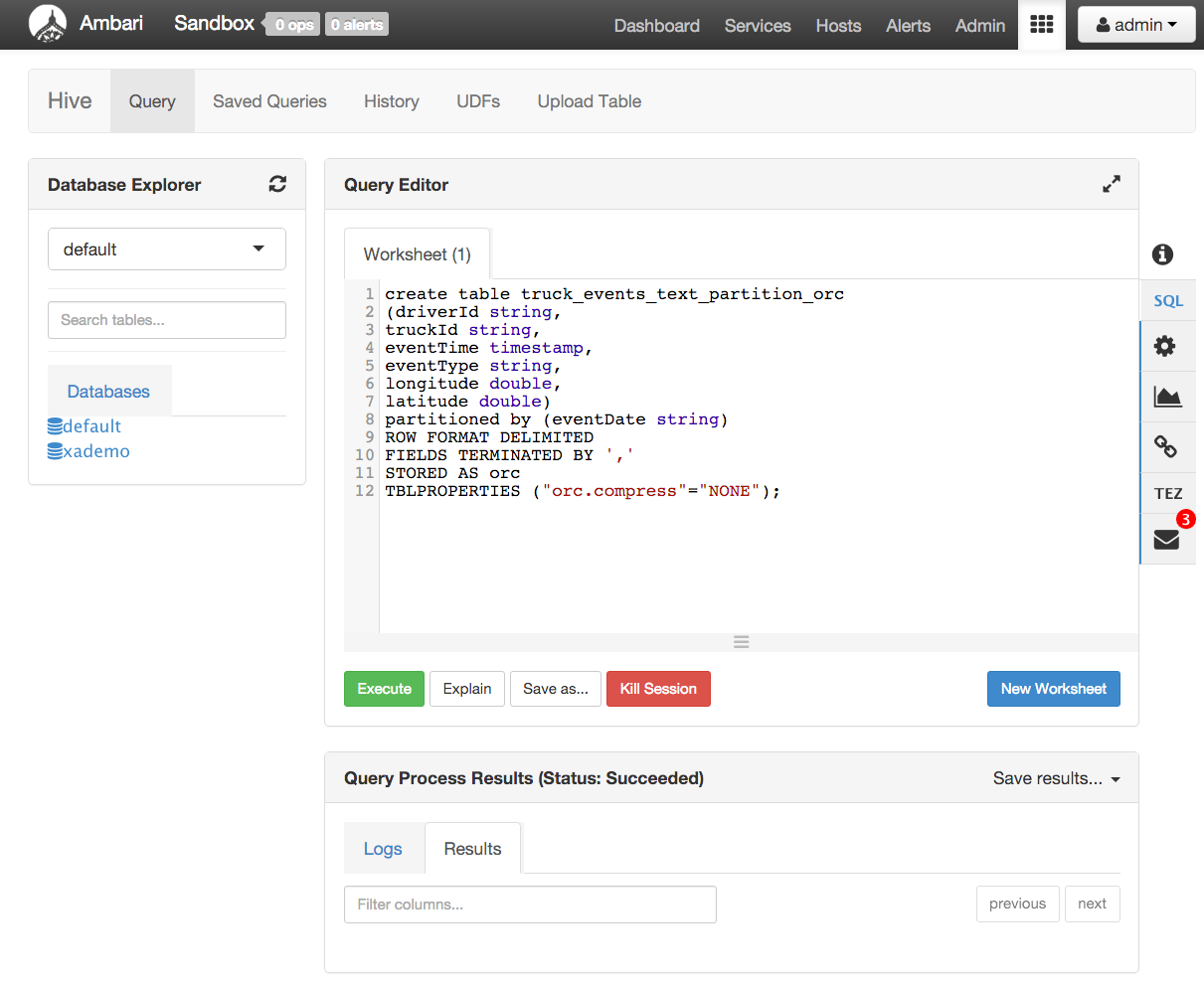
ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS orc

TBLPROPERTIES ("orc.compress"="NONE");

Refresh the Database Explorer and you should see the new table appear under default:



The data in 'truck\_events\_text\_partition\_orc' table can be stored with ZLIB, Snappy, LZO compression options. This can be set by changing tblproperties ("orc.compress"="NONE")option in the query above.

* Set permissions on /tmp/hive

chmod -R 777 /tmp/hive/

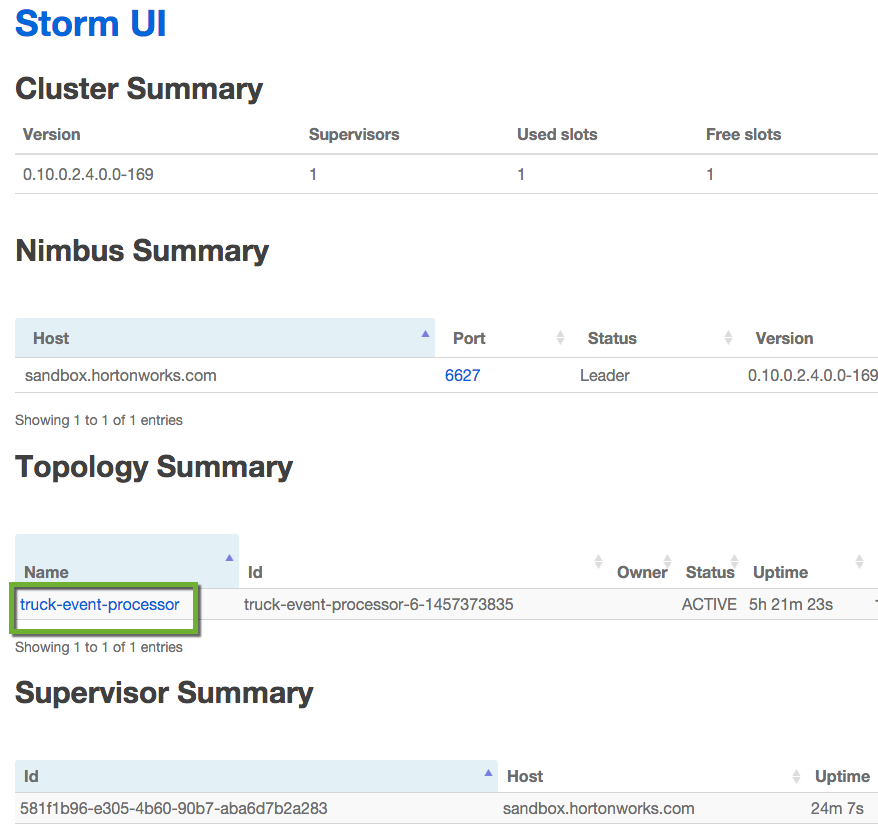
**Step 3: Launch new Storm topology**

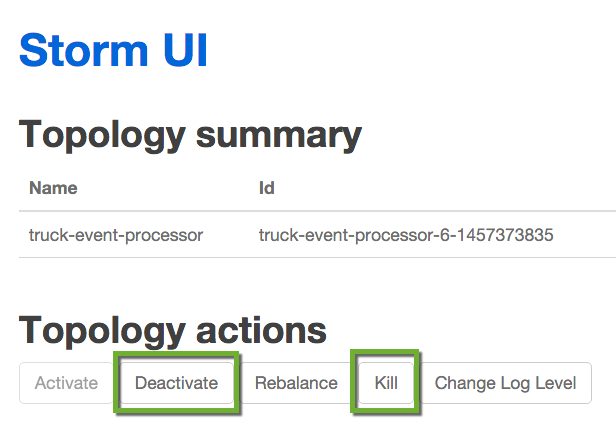
Recall that the source code is under /opt/TruckEvents/Tutorials-master/src directory. The pre-compiled jars are under the /opt/TruckEvents/Tutorials-master/target directory.

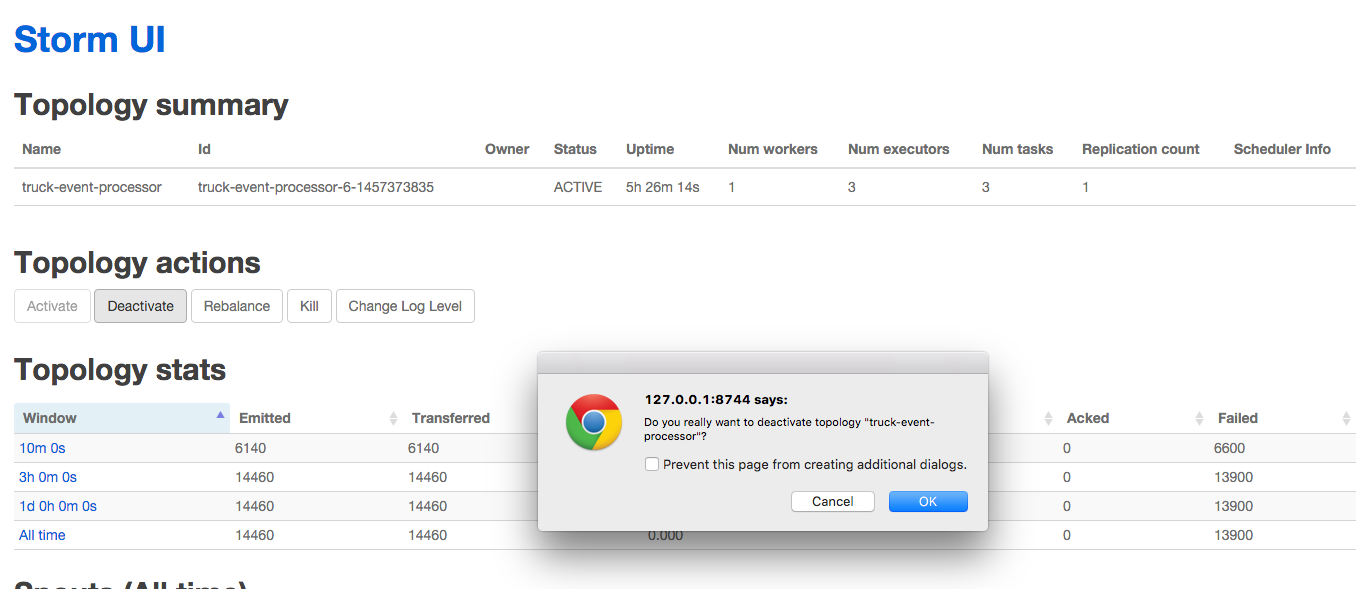
**(Optional)** If you would like to modify/run the code:

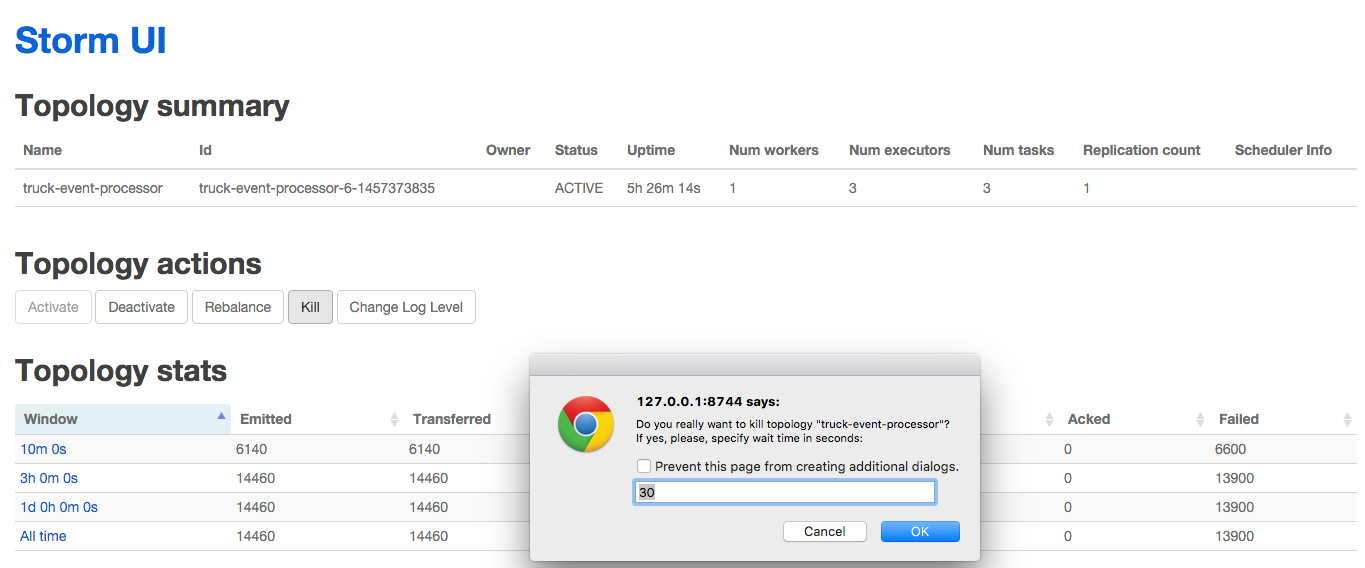
* refer to [Appendix A](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#update-tutorials-master-project) for the steps to run maven to compile the jars to the target subdir from terminal command line
* refer to [Appendix B](http://hortonworks.com/hadoop-tutorial/simulating-transporting-realtime-events-stream-apache-kafka/#enable-remote-desktop-setup-topology) for the steps to enable VNC (i.e. 'remote desktop') access on your sandbox and open/compile the code using Eclipse

If its still running, deactivate or kill the previous Storm topology using the Storm UI as shown in the screenshots below:









The Storm topology can be deactivated/killed from the Storm UI as above or via shell:

storm kill TruckEventProcessingTopology

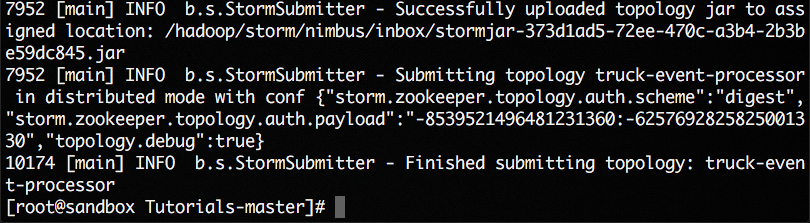
* Load new Storm topology.

Execute the Storm **jar** command to create a new Topology from Tutorial #3 after the code has been compiled.

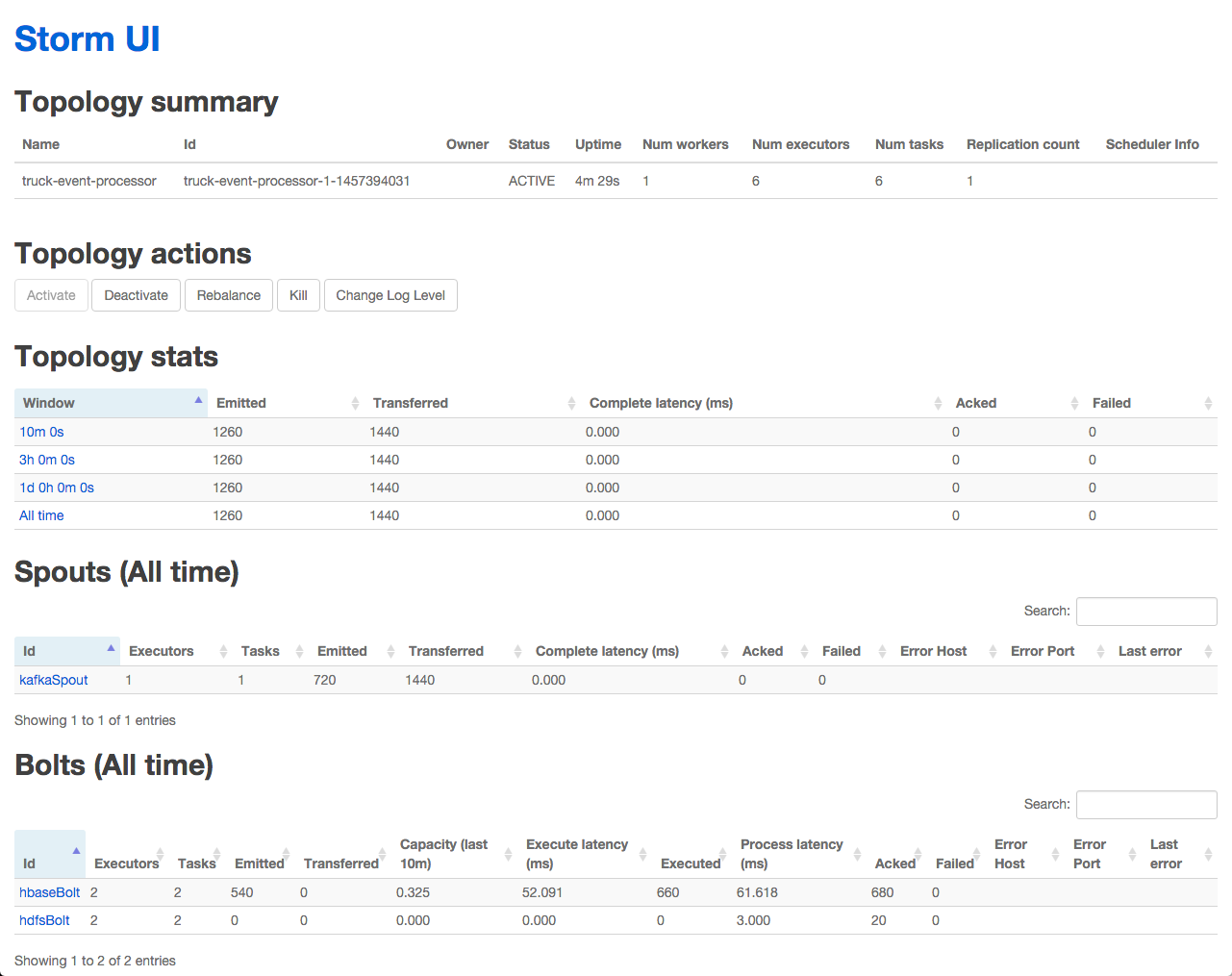
[root@sandbox ~]# cd /opt/TruckEvents/Tutorials-master/

[root@sandbox Tutorials-master]# storm jar target/Tutorial-1.0-SNAPSHOT.jar com.hortonworks.tutorials.tutorial3.TruckEventProcessingTopology

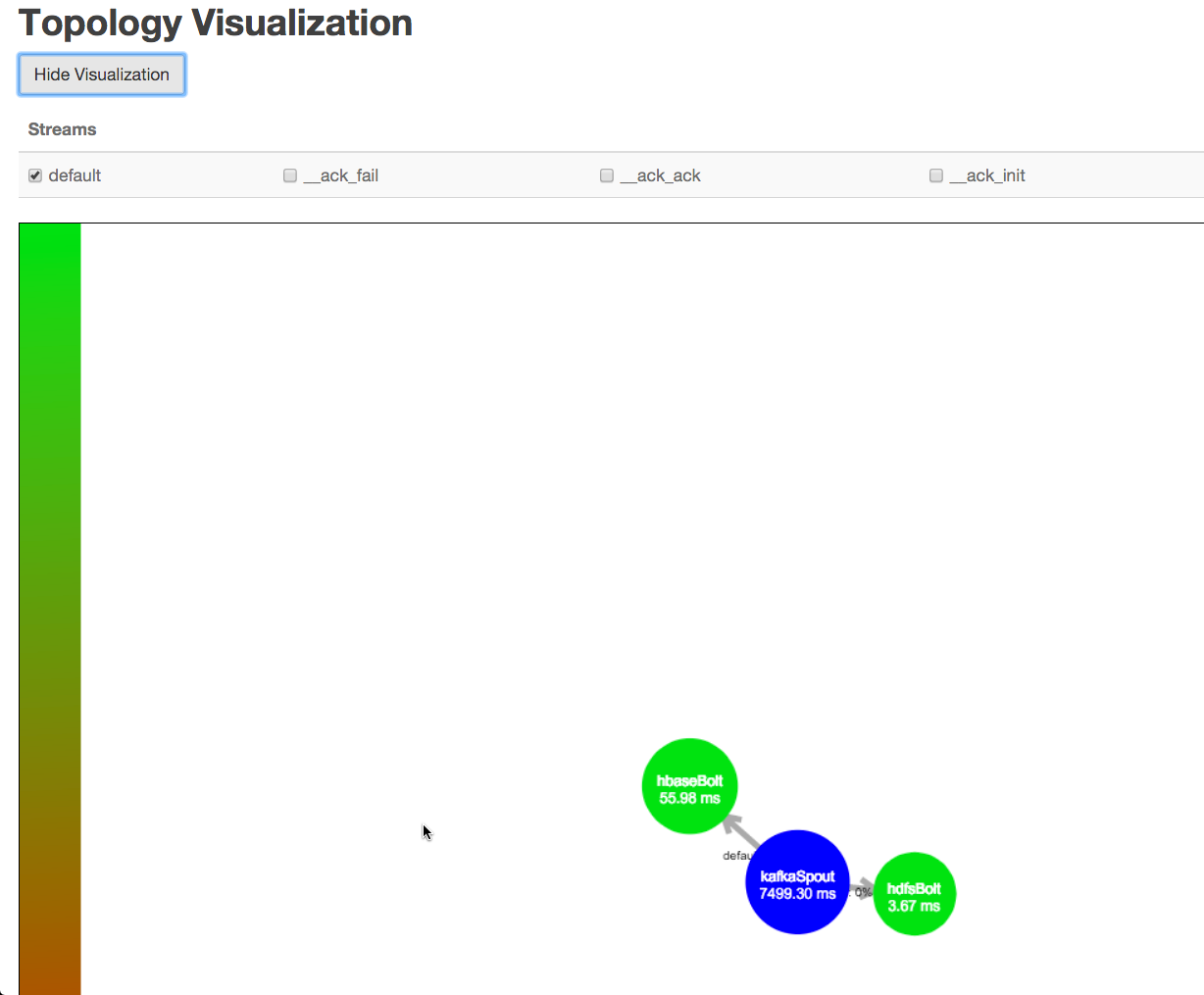
You should see that it successfully submitted the topology:

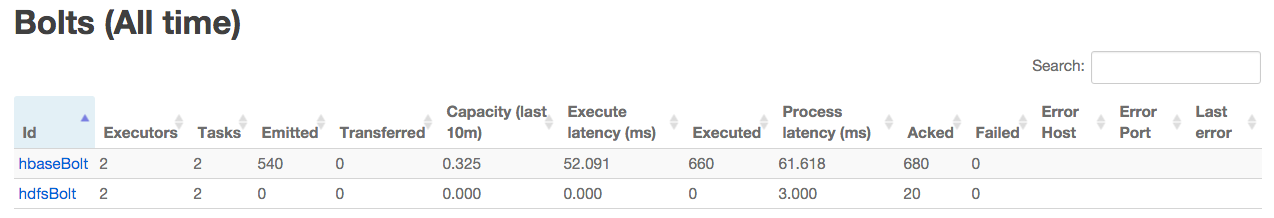


The topology should also show up on the Storm UI as in the image below. After a few minutes, you should see that the number of emitted and transferred tuples is increasing. This increase shows that the messages are process in real time.



Under Topology Visualization: You shall see here that Kafka Spout has started writing to hdfs and hbase.



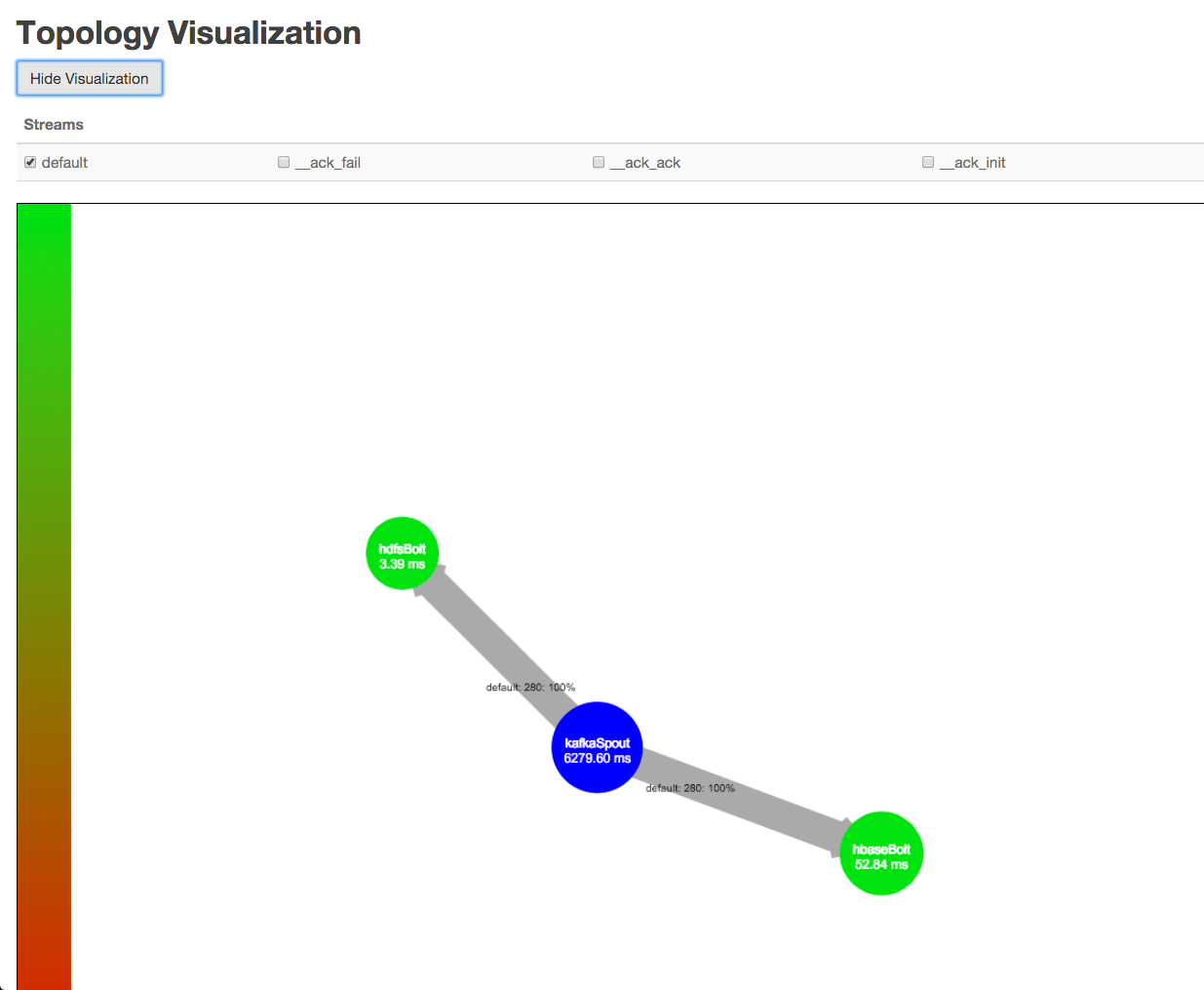


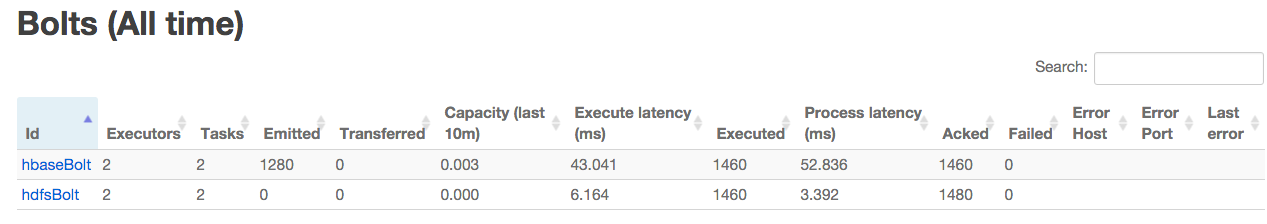
**Step 4: Generate Events and Verify Data in HDFS and HBase**

* Start the 'TruckEventsProducer' Kafka Producer and verify that the data has been persisted by using the Storm Topology view.

[root@sandbox Tutorials-master]# java -cp target/Tutorial-1.0-SNAPSHOT.jar com.hortonworks.tutorials.tutorial1.TruckEventsProducer sandbox.hortonworks.com:6667 sandbox.hortonworks.com:2181

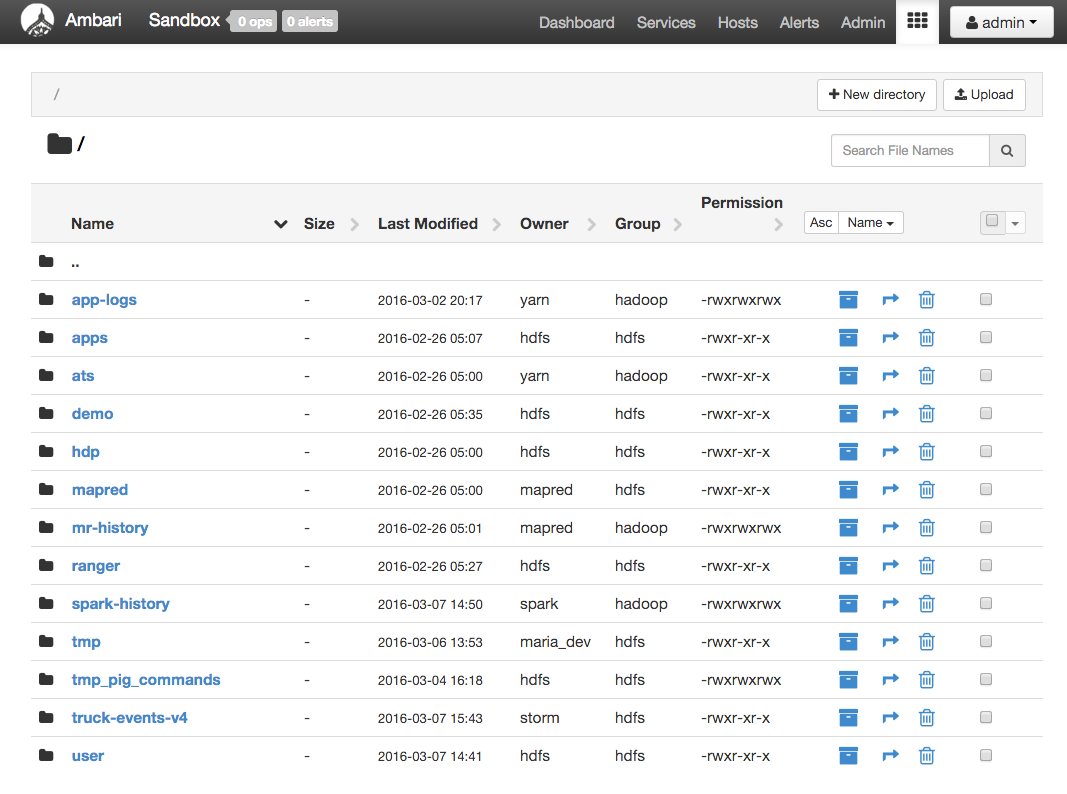
Verify in the Storm UI or Storm User View to verify the Bolt section that HDFS/HBase tuples are being executed and acked



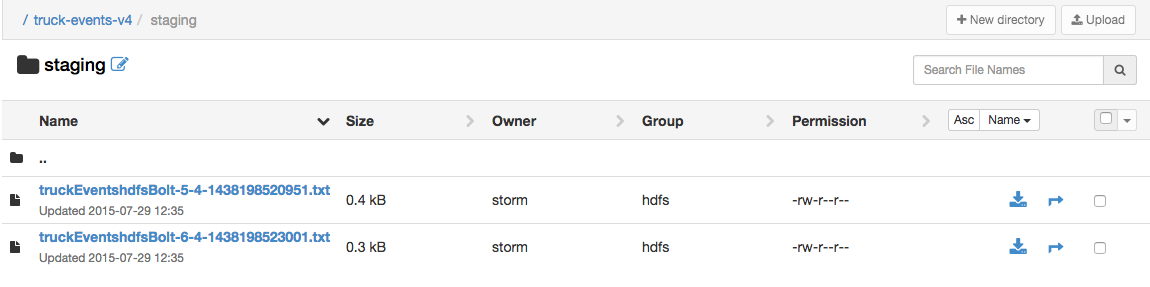


* Verify that the data is in HDFS by opening the Ambari Files view: http://localhost:8080/#/main/views/FILES/0.1.0/MyFiles

With the default settings for HDFS, users will see the data written to HDFS once in every few minutes.



Drill down into /truck-events-v4/staging dir in HDFS

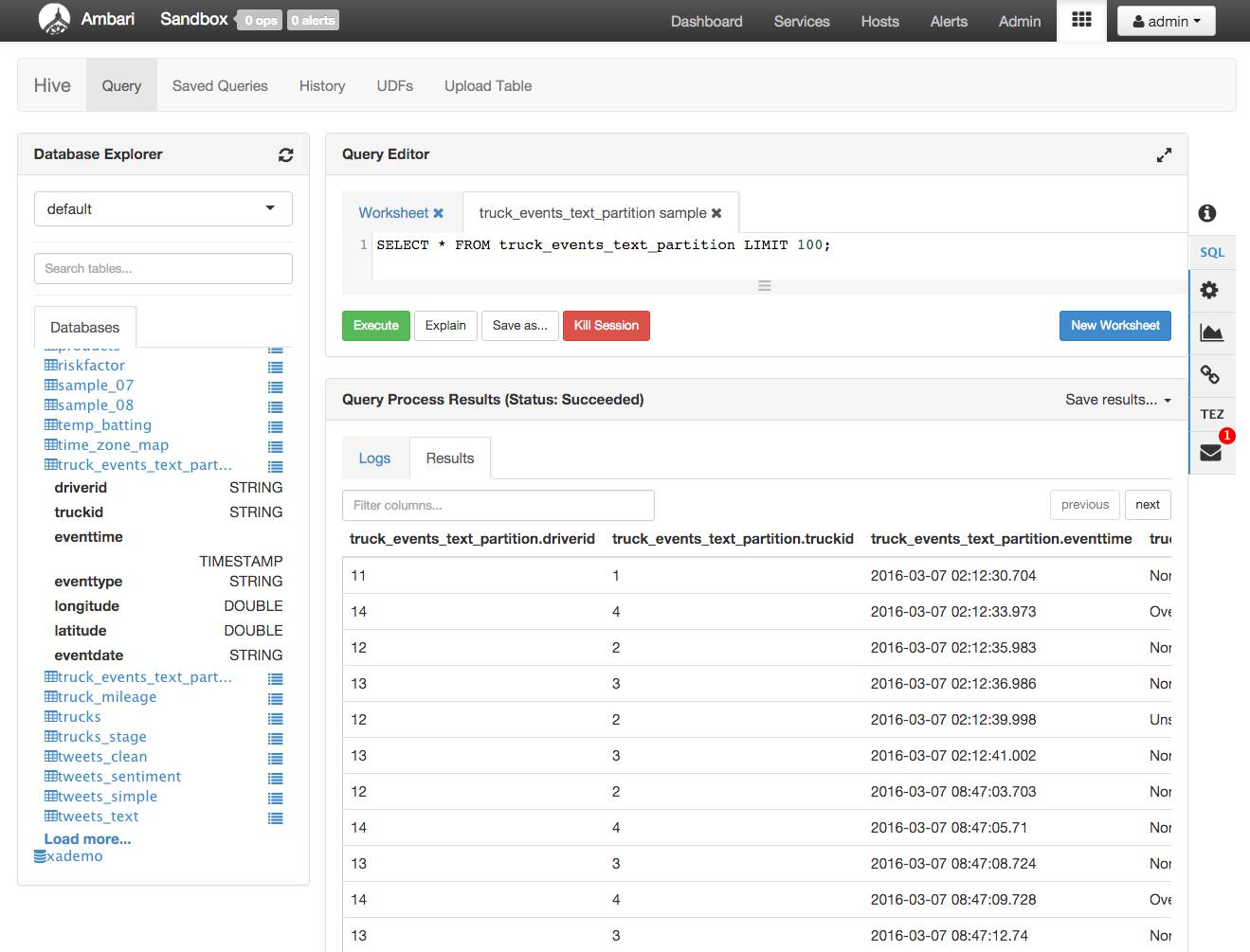


Press Control-C in the terminal to stop the Kafka producer. Now go back to the staging directory, click on one of the txt files and confirm that it contains the events:



**Note:** It may take a 5-10 minutes, before you can access the txt files to see the data.

* Verify data in Hive by navigating to the Hive view, expanding the default database and clicking the List icon next to truck\_events\_text\_partition table



* If you haven't done so, you can press Control-C to stop the Kafka producer (i.e keep Control key pressed and then press C)
* Verify that the data is in HBase by executing the following commands in HBase shell:

[root@sandbox Tutorials-master]# hbase shell

hbase(main):001:0> list

hbase(main):002:0> count 'truck\_events'

366 row(s) in 0.3900 seconds

=> 366

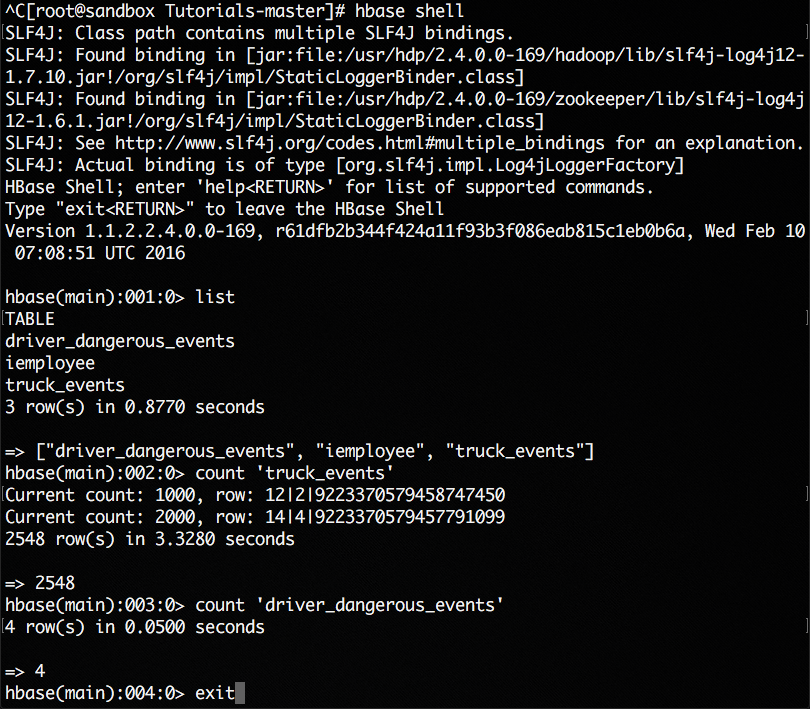
hbase(main):003:0> count 'driver\_dangerous\_events'

3 row(s) in 0.0130 seconds

=> 3

hbase(main):004:0> exit

The driver\_dangerous\_events table is updated upon every violation.

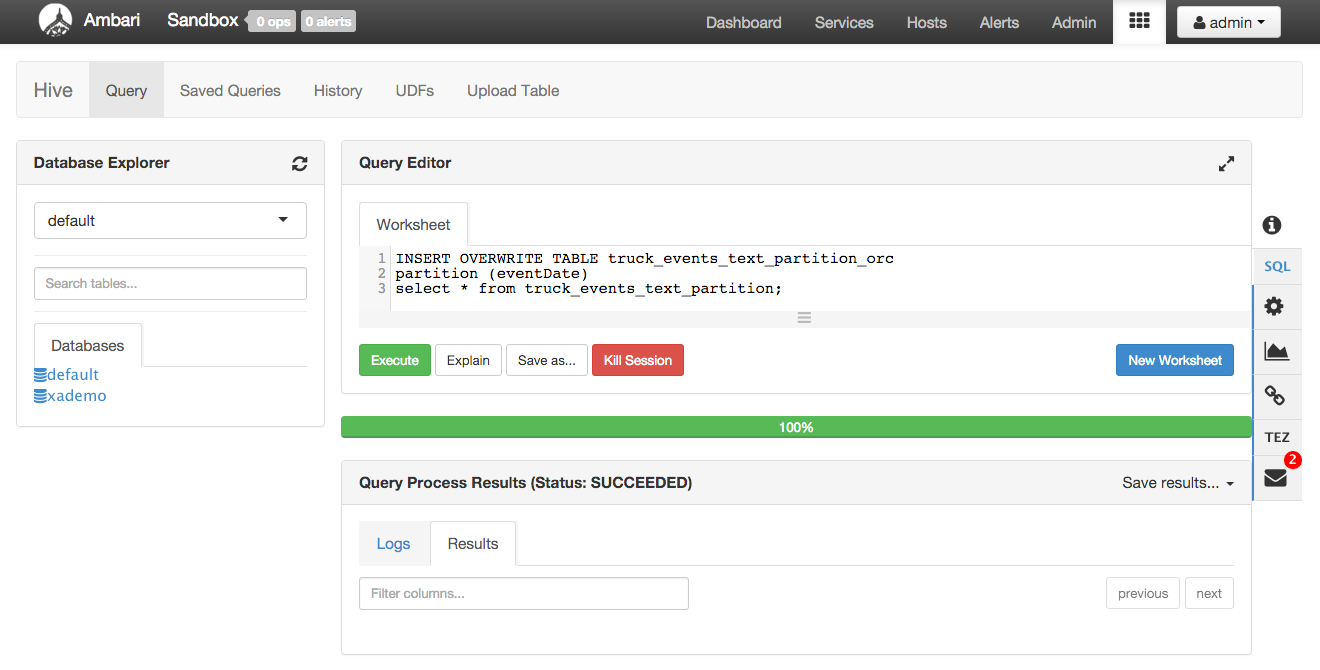


* Next let's populate the data into ORC table for interactive query by Excel (or any BI tool) via ODBC over Hive/Tez. Open the Hive view and enter the below and click Execute.

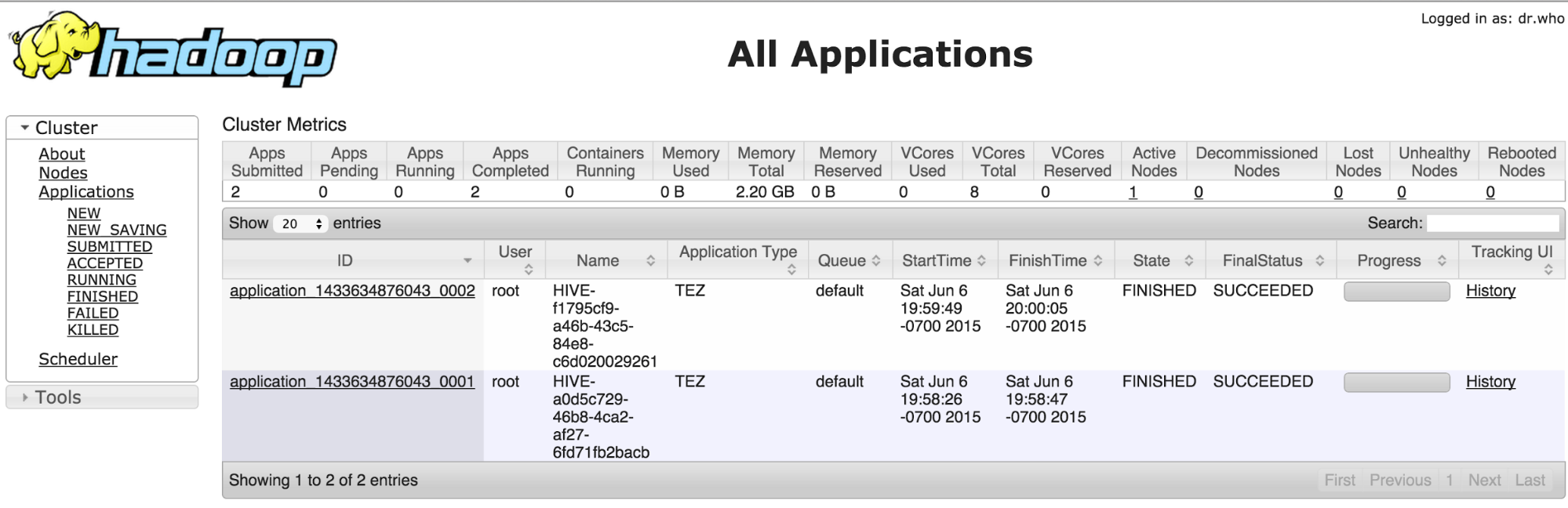
INSERT OVERWRITE TABLE truck\_events\_text\_partition\_orc

partition (eventDate)

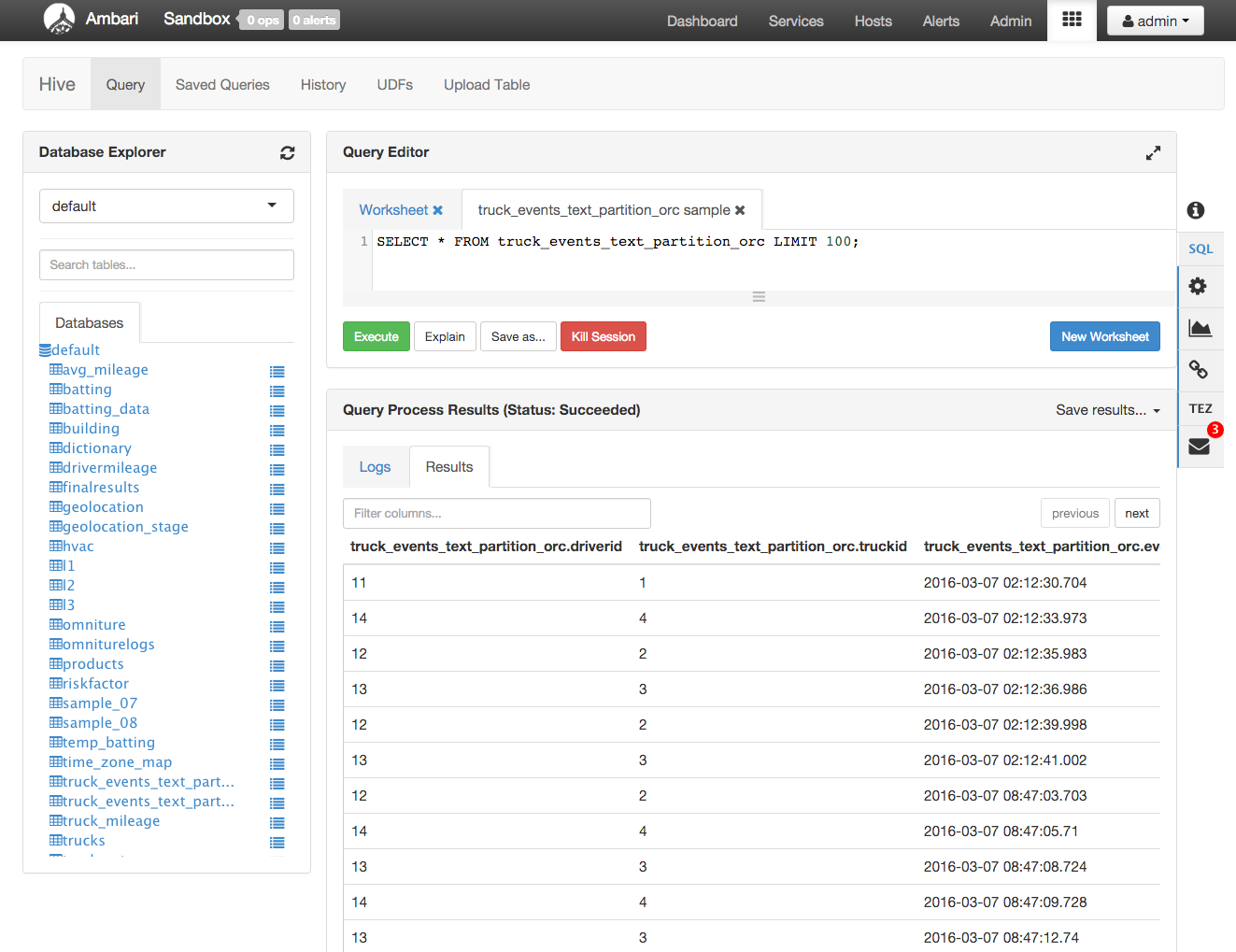
select \* from truck\_events\_text\_partition;



Notice that this launches a Tez job in the background. You can get more details on this using the Yarn resource manager UI. You can find for this under the link under Ambari -> Yarn -> Quick links but will be similar to http://localhost:8088/cluster



Now query the ORC table by clicking the List icon next to it under Databases and notice it is also now populated



* Once done, stop the Storm topology

The Storm topology can be deactivated/killed from the Storm UI or

storm kill TruckEventProcessingTopology

**Conclusion**

This completes the tutorial #3. You have seen how to store streaming data into multiple sources for persistence.

**Step 5: Code Description**

1.BaseTruckEventTopology.java

topologyConfig.load(ClassLoader.getSystemResourceAsStream(configFileLocation));

This is the base class, where the topology configuration is initialized from the /resource/truck\_event\_topology.properties files.

2.FileTimeRotationPolicy.java

This implements the file rotation policy after a certain duration.

public FileTimeRotationPolicy(float count, Units units) {

this.maxMilliSeconds = (long) (count \* units.getMilliSeconds());

}

@Override

public boolean mark(Tuple tuple, long offset) {

// The offsett is not used here as we are rotating based on time

long diff = (new Date()).getTime() - this.lastCheckpoint;

return diff >= this.maxMilliSeconds;

}

3.LogTruckEventsBolt.java

LogTruckEvent Spout logs the Kafka messages received from the Kafka Spout to the log files under /var/log/storm/worker-\*.log

public void execute(Tuple tuple)

{

LOG.info(tuple.getStringByField(TruckScheme.FIELD\_DRIVER\_ID) + "," +

tuple.getStringByField(TruckScheme.FIELD\_TRUCK\_ID) + "," +

tuple.getValueByField(TruckScheme.FIELD\_EVENT\_TIME) + "," +

tuple.getStringByField(TruckScheme.FIELD\_EVENT\_TYPE) + "," +

tuple.getStringByField(TruckScheme.FIELD\_LATITUDE) + "," +

tuple.getStringByField(TruckScheme.FIELD\_LONGITUDE));

}

4.TruckScheme.java

This is the deserializer provided to the Kafka Spout to deserialize Kafka's byte message streams to Values objects.

public List<Object> deserialize(byte[] bytes)

{

try

{

String truckEvent = new String(bytes, "UTF-8");

String[] pieces = truckEvent.split("\\|");

Timestamp eventTime = Timestamp.valueOf(pieces[0]);

String truckId = pieces[1];

String driverId = pieces[2];

String eventType = pieces[3];

String longitude= pieces[4];

String latitude  = pieces[5];

return new Values(cleanup(driverId), cleanup(truckId),

eventTime, cleanup(eventType), cleanup(longitude), cleanup(latitude));

}

catch (UnsupportedEncodingException e)

{

LOG.error(e);

throw new RuntimeException(e);

}

}

5.HiveTablePartitionAction.java

This creates Hive partitions based on timestamp and loads the data by executing the Hive DDL statements.

public void loadData(String path, String datePartitionName, String hourPartitionName )

{

String partitionValue = datePartitionName + "-" + hourPartitionName;

LOG.info("About to add file["+ path + "] to a partitions["+partitionValue + "]");

StringBuilder ddl = new StringBuilder();

ddl.append(" load data inpath ")

.append(" '").append(path).append("' ")

.append(" into table ")

.append(tableName)

.append(" partition ").append(" (date='").append(partitionValue).append("')");

startSessionState(sourceMetastoreUrl);

The data is stored in the partitioned ORC tables using the following method.

String ddlORC = "INSERT OVERWRITE TABLE " + tableName + "\_orc SELECT \* FROM " +tableName;

try {

execHiveDDL("use " + databaseName);

execHiveDDL(ddl.toString());

execHiveDDL(ddlORC.toString());

}

catch (Exception e) {

String errorMessage = "Error exexcuting query["+ddl.toString() + "]";

LOG.error(errorMessage, e);

throw new RuntimeException(errorMessage, e);

}

}

6. TruckEventProcessingTopology.java

This creates a connection to HBase tables and access data within the prepare() function.

public void prepare(Map stormConf, TopologyContext context, OutputCollector collector)

{

...

this.connection = HConnectionManager.createConnection(constructConfiguration());

this.eventsCountTable = connection.getTable(EVENTS\_COUNT\_TABLE\_NAME);

this.eventsTable = connection.getTable(EVENTS\_TABLE\_NAME);

}

...

}

Data to be stored is prepared in the constructRow() function using put.add().

private Put constructRow(String columnFamily, String driverId, String truckId,

Timestamp eventTime, String eventType, String latitude, String longitude)

{

String rowKey = consructKey(driverId, truckId, eventTime);

...

put.add(CF\_EVENTS\_TABLE, COL\_DRIVER\_ID, Bytes.toBytes(driverId));

put.add(CF\_EVENTS\_TABLE, COL\_TRUCK\_ID, Bytes.toBytes(truckId));

...

}

This executes the getInfractionCountForDriver() to get the count of events for a driver using driverID and stores the data in HBase with constructRow() function.

public void execute(Tuple tuple)

{

...

long incidentTotalCount = getInfractionCountForDriver(driverId);

...

Put put = constructRow(EVENTS\_TABLE\_NAME, driverId, truckId, eventTime, eventType,

latitude, longitude);

this.eventsTable.put(put);

...

incidentTotalCount = this.eventsCountTable.incrementColumnValue(Bytes.toBytes(driverId), CF\_EVENTS\_COUNT\_TABLE,

...

}

7. TruckEventProcessingTopology.java

HDFS and HBase Bolt configurations created within configureHDFSBolt() and configureHBaseBolt() respectively.

public void configureHDFSBolt(TopologyBuilder builder)

{

HdfsBolt hdfsBolt = new HdfsBolt()

.withFsUrl(fsUrl)

.withFileNameFormat(fileNameFormat)

.withRecordFormat(format)

.withRotationPolicy(rotationPolicy)

.withSyncPolicy(syncPolicy)

.addRotationAction(hivePartitionAction);

}

public void configureHBaseBolt(TopologyBuilder builder)

{

TruckHBaseBolt hbaseBolt = new TruckHBaseBolt(topologyConfig);

builder.setBolt(HBASE\_BOLT\_ID, hbaseBolt, 2).shuffleGrouping(KAFKA\_SPOUT\_ID);

}

**Appendix A: Update Tutorials-master Project**

* Copy /etc/hbase/conf/hbase-site.xml to src/main/resources/ directory

[root@sandbox ~]# cd /opt/TruckEvents/Tutorials-master

[root@sandbox ~]# cp /etc/hbase/conf/hbase-site.xml src/main/resources/

* Check pom.xml to ensure it includes the below dependencies (check after **line 104**)

<dependency>

<groupId>xerces</groupId>

<artifactId>xercesImpl</artifactId>

<version>2.9.1</version>

</dependency>

<dependency>

<groupId>xalan</groupId>

<artifactId>xalan</artifactId>

<version>2.7.1</version>

</dependency>

<dependency>

<groupId>org.htrace</groupId>

<artifactId>htrace-core</artifactId>

<version>3.0.4</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

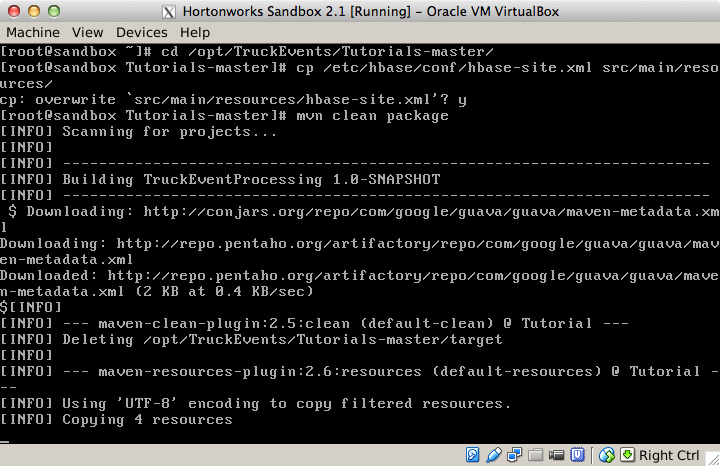
<artifactId>hadoop-hdfs</artifactId>

<version>2.6.0</version>

</dependency>

* recompile the Maven project. This may run for 10+ min

[root@sandbox ~]# mvn clean package



The maven build should succeed

**Appendix B: Enable remote desktop on sandbox and set up Storm topology as Eclipse project**

1. Setup Ambari VNC service on the sandbox to enable remote desktop via VNC and install eclipse using steps here [https://github.com/hortonworks-gallery/ambari-vnc-service#setup-vnc-service](https://github.com/hortonworks-gallery/ambari-vnc-service%23setup-vnc-service)
2. Import code as Eclipse project using steps here:

[https://github.com/hortonworks-gallery/ambari-vnc-service#getting-started-with-storm-and-maven-in-eclipse-environment](https://github.com/hortonworks-gallery/ambari-vnc-service%23getting-started-with-storm-and-maven-in-eclipse-environment)