Exercise #1:

REAL TIME DATA INGESTION IN HBASE AND HIVE USING STORM

INTRODUCTION

The Trucking business is a high-risk business in which truck drivers venture into remote areas, often in harsh weather conditions and chaotic traffic on a daily basis. Using this solution illustrating Modern Data Architecture with Hortonworks Data Platform, we have developed a centralized management system that can help reduce risk and lower the total cost of operations.

This system can take into consideration adverse weather conditions, the driver's driving patterns, current traffic conditions and other criteria to alert and inform the management staff and the drivers themselves when risk factors run high.

In previous exercise, we have explored generating and capturing streaming data with Apache NiFi and Apache Kafka.

In this exercise, we will build a solution to ingest real time streaming data into HBase using Storm. Storm has a spout that reads truck_events data from Kafka and passes it to bolts, which process and persist the data into Hive & HBase tables.

PRE-REQUISITES

- Day 1 Exercise 3: Set Up Simulator, Apache Services and IDE Environment
- Day 2 Exercise 1: Ingest, Route and Land Real Time Events with Apache NiFi
- Day 2 Exercise 2: Capture Real Time Events with Apache Kafka

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.
It's used for:

- Real time analytics
- Scoring machine learning models

- Continuous statics computations
- Operational Analytics
- And, to enforce Extract, Transform, and Load (ETL) paradigms.

  A Storm Topology is network of Spouts and Bolts. The Spouts generate streams, which contain sequences of tuples (data) while the Bolts process input streams and produce output streams. Hence, the Storm Topology can talk to databases, run functions, filter, merge or join data. We will be using Storm parse data, perform complex computations on truck events and send data to HBase Tables.

- **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. We have a RouteBolt that transforms the tuple and passes that data onto the other for further processing. We have 3 HBase bolts that write to 3 tables.

  Learn more about Apache Storm at the Storm Documentation page.

EXERCISE OVERVIEW

- Create HBase Tables
- Deploy Storm Topology
- Analyze a Storm Spout and several Bolts.
- Store Persisting data into HBase.
- Verify Data Stored in HBase.

STEP 1: CREATE TABLES IN HBASE

- Create HBase tables

  We will work with 3 Hbase tables in this exercise.

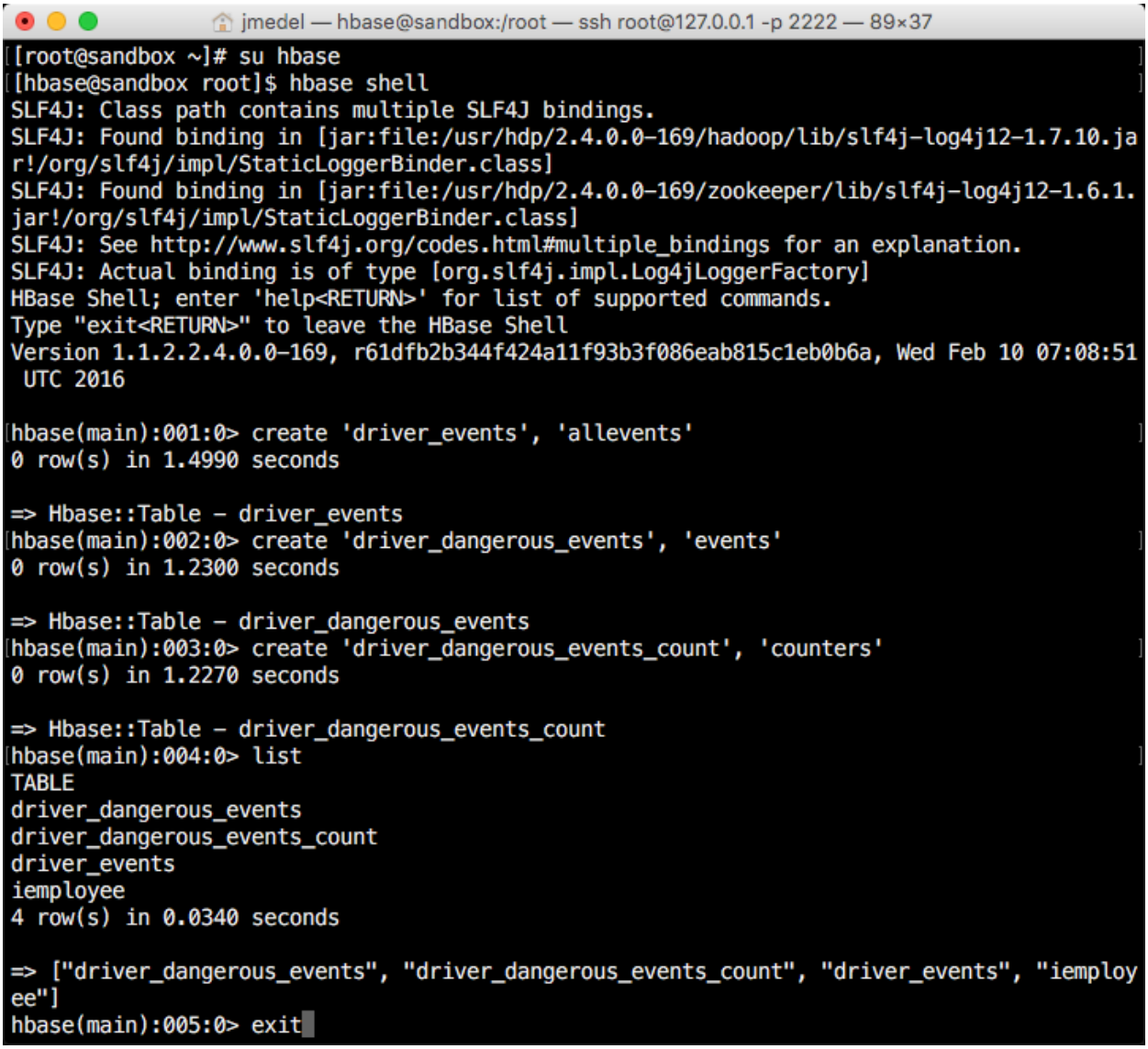
  The first table stores **all events** generated, the second stores only **dangerous events** and third stores the **number of incidents per driverId**.

```
su hbase

hbase shell

create 'driver_events', 'allevents'
create 'driver_dangerous_events', 'events'
create 'driver_dangerous_events_count', 'counters'
list
exit
```

Now let's exit from hbase user, type `exit`.

- **driver_events** can be thought of as **All Events** table
- **driver_dangerous_events** can be thought of as **Dangerous Events** Table
- **driver_dangerous_events_count** can be thought of as **Incidents Per Driver** Table

  Note: 'driver_events' is the table name and 'allevents' is column family.
  In the script above, we have one column family. Yet, if we want we can have multiple column families. We just need to include more arguments.



STEP 2: LAUNCH STORM TOPOLOGY

Recall that the source code is under directory path `iot-truck-streaming/storm-streaming/src/` .

The pre-compiled jars are under the directory path `iot-truck-streaming/storm-streaming/target/` .

Note: Back in Day 1 Exercise 3 in which we set up the Trucking Demo, we used maven to create the target folder.

2.1 VERIFY KAFKA IS RUNNING & DEPLOY TOPOLOGY

1. Verify that Kafka service is running using Ambari dashboard. If not, start the Kafka service as we did in the last exercise.
2. Deploy Storm Topology

We now have 'supervisor' daemon and Kafka processes running.
To do real-time computation on Storm, you create what are called "topologies". A topology is a Directed Acyclic Graph (DAG) of spouts and bolts with streams of tuples representing the edges. Each node in a topology contains processing logic, and links between nodes indicate how data should be passed around between nodes.

Running a topology is straightforward. First, you package all your code and dependencies into a single jar. In exercise 0 when we set up our IDE environment, we ran mvn clean package after we were satisfied with the state of our code for the topology, which packaged our storm project into a **storm…SNAPSHOT.jar**. The command below will deploy a new Storm Topology for Truck Events.

```
[root@sandbox ~]# cd ~/iot-truck-streaming
[root@sandbox iot-truck-streaming]# storm jar storm-streaming/target/storm-streaming-
1.0-SNAPSHOT.jar
com.hortonworks.streaming.impl.topologies.TruckEventKafkaExperimTopology
/etc/storm_demo/config.properties
```

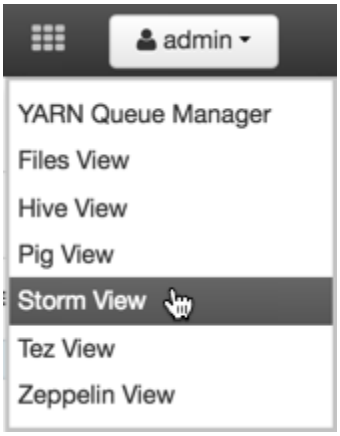
You should see that the topology deployed successfully:

```
9276 [main] INFO  b.s.StormSubmitter - Finished submitting topology: truck-event-processor
[root@sandbox iot-truck-streaming]#
```

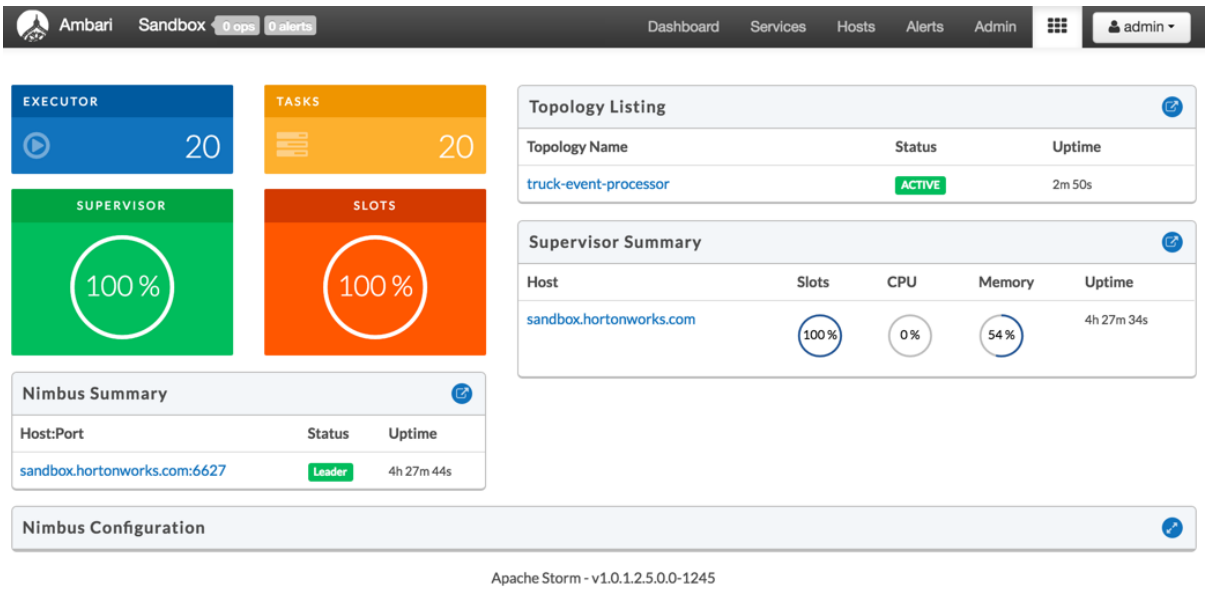
This runs the class **TruckEventKafkaExperimTopology**. The main function of the class defines the topology and submits it to Nimbus. The storm jar part takes care of connecting to Nimbus and uploading the jar.

Open your Ambari Dashboard. Click the **Storm View** located in the Ambari

User Views list.

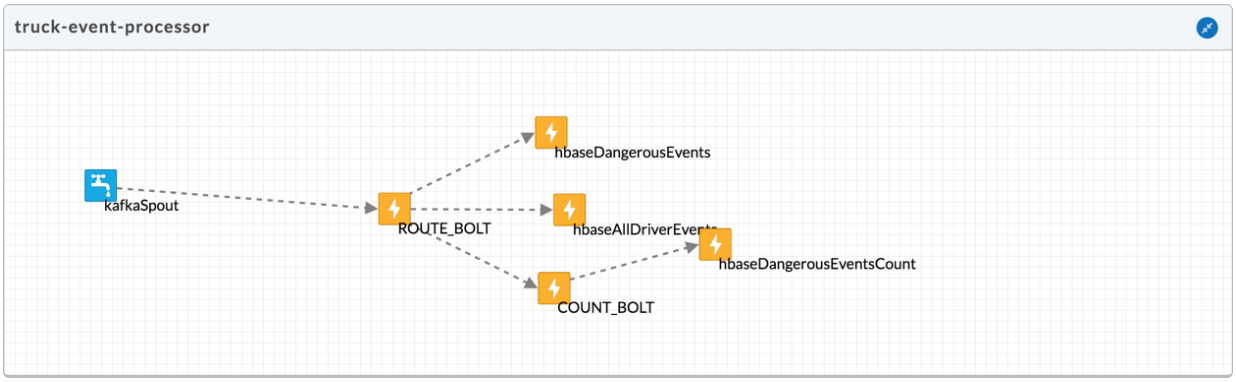


You should see the new Topology **truck-event-processor**.



Run the NiFi DataFlow to generate events.
Return to the Storm View and click on **truck-event-processor** topology in the list of topologies to drill into it.

As you scroll down the page, let's analyze a Visualization of our truck-event-processor topology:



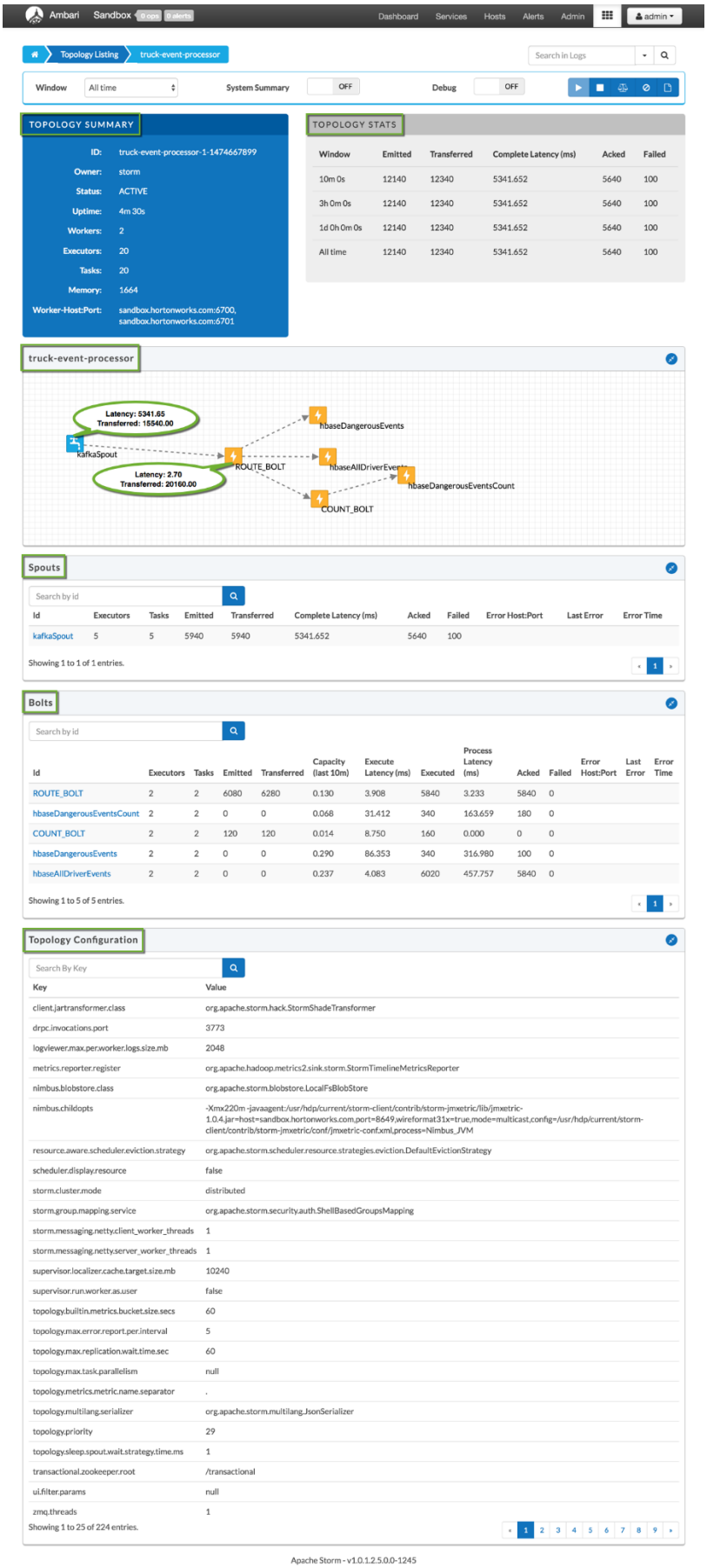
2.2 ANALYSIS OF TOPOLOGY VISUALIZATION:

- RouteBolt processes the data received by KafkaSpout
- CountBolt takes the data from RoutBolt and counts the incidents per driver
- 1 HBaseBolt performs complex transformations on the data received by CountBolt to write to **Incidents Per Driver** Table
- 2 HBase Bolts perform complex transformations on the data from RouteBolt to write to **All Events** and **Dangerous Event** Tables.

2.3 OVERVIEW OF THE STORM VIEW

After 6-10 minutes, you should see that numbers of emitted and transferred tuples for each node (Spout or Bolt) in the topology is increasing, which shows that the messages are processed in real time by Spout and Bolts. If we hover over one of the spouts or bolts, we can see how much data they process and their latency.

Here is an example of the data that goes through the kafkaSpout and RouteBolt in the topology:


Ambari
Sandbox
Dashboard
Services
Hosts
Alerts
Admin
admin
Topology Listing
truck-event-processor
Search in Logs
Window
All time
System Summary
OFF
Debug
OFF
TOPOLOGY SUMMARY
ID: truck-event-processor-1-1474667899
Owner: storm
Status: ACTIVE
Uptime: 4m 30s
Workers: 2
Executors: 20
Tasks: 20
Memory: 1664
Worker-Host:Port: sandbox.hortonworks.com:6700, sandbox.hortonworks.com:6701
TOPOLOGY STATS
Window | Emitted | Transferred | Complete Latency (ms) | Acked | Failed
10m 0s | 12140 | 12340 | 5341.652 | 5640 | 100
3h 0m 0s | 12140 | 12340 | 5341.652 | 5640 | 100
1d 0h 0m 0s | 12140 | 12340 | 5341.652 | 5640 | 100
All time | 12140 | 12340 | 5341.652 | 5640 | 100
truck-event-processor
Latency: 5341.65
Transferred: 15540.00
kafkaSpout
Latency: 2.70
Transferred: 20160.00
ROUTE_BOLT
hbaseDangerousEvents
hbaseAllDriverEvents
hbaseDangerousEventsCount
COUNT_BOLT
Spouts
Search by id
Id | Executors | Tasks | Emitted | Transferred | Complete Latency (ms) | Acked | Failed | Error Host:Port | Last Error | Error Time
kafkaSpout | 5 | 5 | 5940 | 5940 | 5341.652 | 5640 | 100
Showing 1 to 1 of 1 entries.
Bolts
Search by id
Id | Executors | Tasks | Emitted | Transferred | Capacity (last 10m) | Execute Latency (ms) | Executed | Process Latency (ms) | Acked | Failed | Error Host:Port | Last Error | Error Time
ROUTE_BOLT | 2 | 2 | 6080 | 6280 | 0.130 | 3.908 | 5840 | 3.233 | 5840 | 0
hbaseDangerousEventsCount | 2 | 2 | 0 | 0 | 0.068 | 31.412 | 340 | 163.659 | 180 | 0
COUNT_BOLT | 2 | 2 | 120 | 120 | 0.014 | 8.750 | 160 | 0.000 | 0 | 0
hbaseDangerousEvents | 2 | 2 | 0 | 0 | 0.290 | 86.353 | 340 | 316.980 | 100 | 0
hbaseAllDriverEvents | 2 | 2 | 0 | 0 | 0.237 | 4.083 | 6020 | 457.757 | 5840 | 0
Showing 1 to 5 of 5 entries.
Topology Configuration
Search By Key
Key | Value
client.jartransformer.class | org.apache.storm.hack.StormShadeTransformer
drpc.invocations.port | 3773
logviewer.max.per.worker.logs.size.mb | 2048
metrics.reporter.register | org.apache.hadoop.metrics2.sink.storm.StormTimelineMetricsReporter
nimbus.blobstore.class | org.apache.storm.blobstore.LocalFsBlobStore
nimbus.childopts | -Xmx220m -javaagent:/usr/hdp/current/storm-client/contrib/storm-jmxetric/lib/jmxetric-1.0.4.jar=host=sandbox.hortonworks.com,port=8649,wireformat31x=true,mode=multicast,config=/usr/hdp/current/storm-client/contrib/storm-jmxetric/conf/jmxetric-conf.xml,process=Nimbus_JVM
resource.aware.scheduler.eviction.strategy | org.apache.storm.scheduler.resource.strategies.eviction.DefaultEvictionStrategy
scheduler.display.resource | false
storm.cluster.mode | distributed
storm.group.mapping.service | org.apache.storm.security.auth.ShellBasedGroupsMapping
storm.messaging.netty.client_worker_threads | 1
storm.messaging.netty.server_worker_threads | 1
supervisor.localizer.cache.target.size.mb | 10240
supervisor.run.worker.as.user | false
topology.builtin.metrics.bucket.size.secs | 60
topology.max.error.report.per.interval | 5
topology.max.replication.wait.time.sec | 60
topology.max.task.parallelism | null
topology.metrics.metric.name.separator | .
topology.multilang.serializer | org.apache.storm.multilang.JsonSerializer
topology.priority | 29
topology.sleep.spout.wait.strategy.time.ms | 1
transactional.zookeeper.root | /transactional
ui.filter.params | null
zmq.threads | 1
Showing 1 to 25 of 224 entries.
Apache Storm - v1.0.1.2.5.0.0-1245
Licensed under the Apache License, Version 2.0.
See third-party tools/resources that Ambari uses and their respective authors

Overview of truck-event-processor in Storm View

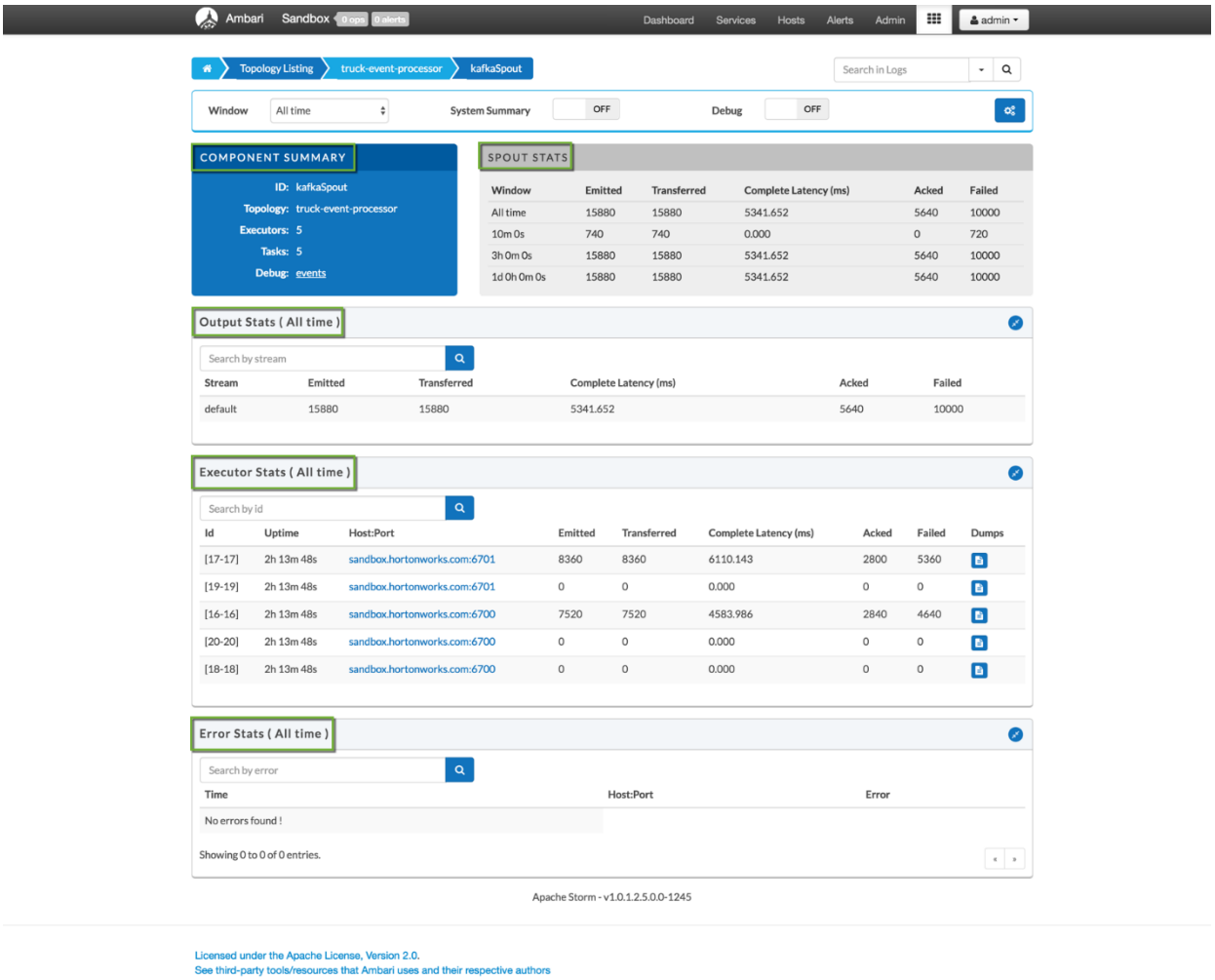
- Topology Summary
- Topology Stats
- truck-event-processor Visualization
- Spout
- Bolts
- Topology Configuration

2.4 OVERVIEW OF SPOUT STATISTICS:

To see statistics of a particular component or node in the storm topology, click on that component located in the Spouts or Bolts section. For instance, let's dive into the KafkaSpout's statistics.

Overview of Spout Statistics

- Component Summary
- Spout Stats
- Output Stats (All time)
- Executor Stats (All time)
- Error Stats (All time)

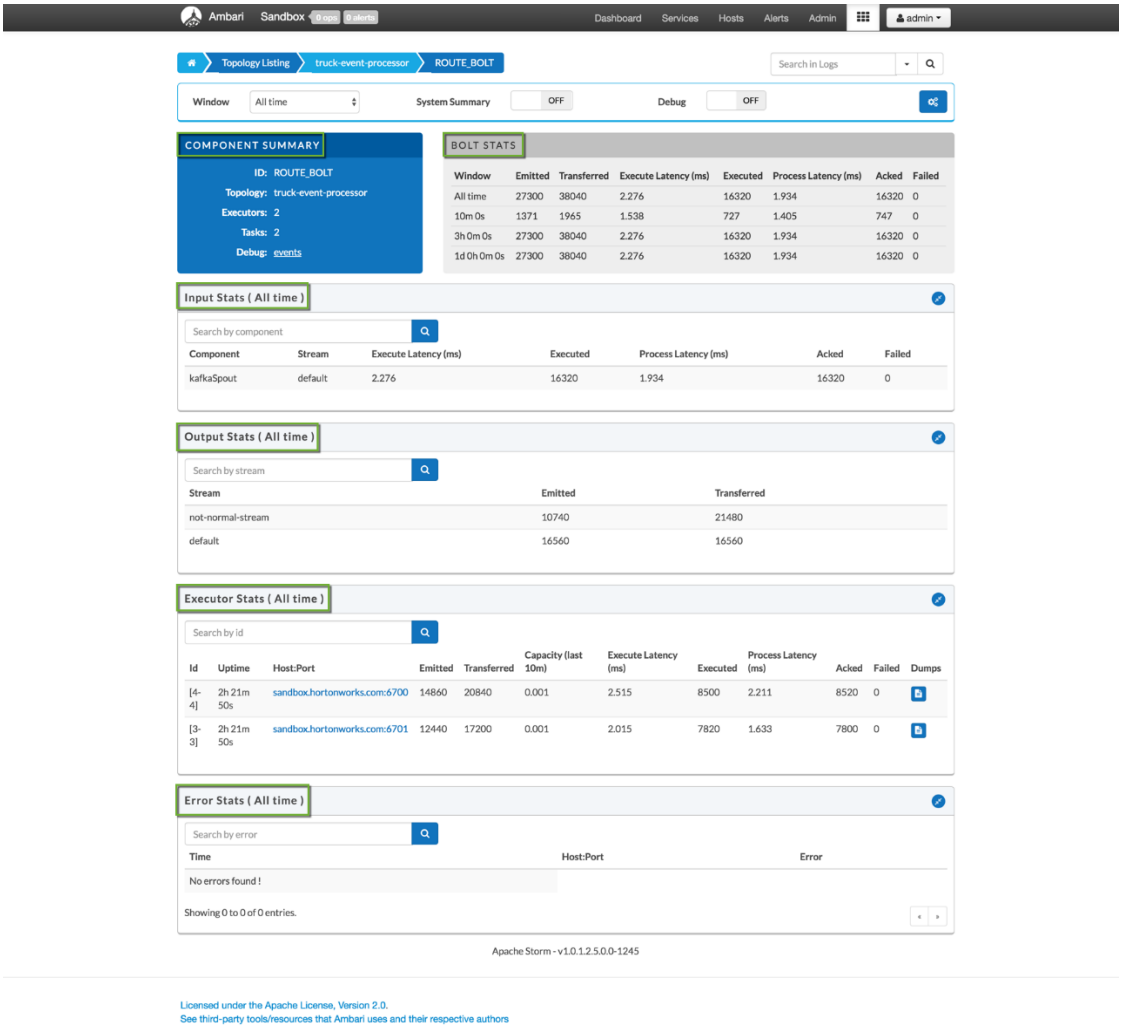


2.5 OVERVIEW OF BOLT STATISTICS:

Follow similar process in section 2.4 to see the statistics of a particular bolt in your topology. Let's dive into the RouteBolt statistics.

Overview of Bolt Statistics

- Component Summary
- Bolt Stats
- Input Stats (All time)
- Output Stats (All time)
- Executor Stats (All time)
- Error Stats (All time)



What differences do you notice about the spout statistics compared to the bolt statistics?

STEP 3: VERIFY DATA IN HBASE

Let's verify that Storm's 3 HBase bolts successfully sent data to the 3 HBase Tables.

- If you haven't done so, you can you can stop the NiFi DataFlow. Press the stop symbol.
- Verify that the data is in HBase by executing the following commands in HBase shell:

```
hbase shell

list
count 'driver_events'
count 'driver_dangerous_events'
count 'driver_dangerous_events_count'
exit
```

The `driver_dangerous_events` table is updated upon every violation event.

```
hbase(main):002:0> scan 'driver_dangerous_events'
ROW                    COLUMN+CELL
 10|58|922337056218683414 column=events:driverId, timestamp=1474668136716, value=\x00\x00\x00\x0A
 5
 10|58|922337056218683414 column=events:driverName, timestamp=1474668136716, value=George Vetticad
 5                       en
 10|58|922337056218683414 column=events:eventType, timestamp=1474668136716, value=Unsafe following
 5                        distance
 10|58|922337056218683414 column=events:hbaseRowKey, timestamp=1474668136716, value=10|58|92233705
 5                       62186834145
 10|58|922337056218683414 column=events:latitude, timestamp=1474668136716, value=@B\x82\x8F\x5C(\x
 5                       F5\xC3
 10|58|922337056218683414 column=events:longitude, timestamp=1474668136716, value=\xC0W\xA2\x8F\x5
 5                       C(\xF5\xC3
 10|58|922337056218683414 column=events:routeId, timestamp=1474668136716, value=R\xDFf\x97
 5
 10|58|922337056218683414 column=events:routeName, timestamp=1474668136716, value=Saint Louis to T
 5                       ulsa
 10|58|922337056218683414 column=events:truckId, timestamp=1474668136716, value=\x00\x00\x00:
 5
 10|58|922337056218683414 column=events:ts, timestamp=1474668136716, value=\x00\x00\x01WY\x0F\x0F\
 5                       x1E
 10|58|922337056218685984 column=events:driverId, timestamp=1474668136523, value=\x00\x00\x00\x0A
 3
 10|58|922337056218685984 column=events:driverName, timestamp=1474668136523, value=George Vetticad
 3                       en
 10|58|922337056218685984 column=events:eventType, timestamp=1474668136523, value=Lane Departure
 3
 10|58|922337056218685984 column=events:hbaseRowKey, timestamp=1474668136523, value=10|58|92233705
 3                       62186859843
 10|58|922337056218685984 column=events:latitude, timestamp=1474668136523, value=@B\xF8Q\xEB\x85\x
 3                       1E\xB8
 10|58|922337056218685984 column=events:longitude, timestamp=1474668136523, value=\xC0V\xFF\x5C(\x
 3                       F5\xC2\x8F
 10|58|922337056218685984 column=events:routeId, timestamp=1474668136523, value=R\xDFf\x97
 3
 10|58|922337056218685984 column=events:routeName, timestamp=1474668136523, value=Saint Louis to T
 3                       ulsa
 10|58|922337056218685984 column=events:truckId, timestamp=1474668136523, value=\x00\x00\x00:
 3
 10|58|922337056218685984 column=events:ts, timestamp=1474668136523, value=\x00\x00\x01WY\x0E\xAA\
 3                       xBC
 10|58|922337056218688633 column=events:driverId, timestamp=1474668137719, value=\x00\x00\x00\x0A
 5
 10|58|922337056218688633 column=events:driverName, timestamp=1474668137719, value=George Vetticad
 5                       en
 10|58|922337056218688633 column=events:eventType, timestamp=1474668137719, value=Lane Departure
 5
```

3.1 TROUBLESHOOT UNEXPECTED DATA IN HBASE TABLE

If the data in the HBase table is displayed in hexadecimal values, you can perform the following special operation on a table to display the data in the correct format. For instance, if your **driver_dangerous_events** had unexpected data, run the following hbase query:

```
scan 'driver_dangerous_events_count' , {COLUMNS => ['counters:driverId:toInt',
'counters:incidentTotalCount:toLong']}
```

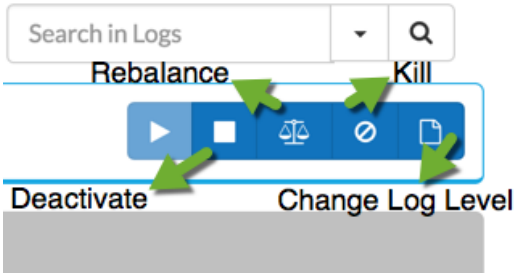
Your data should look as follows:

```
jmedel — hbase@sandbox:/var/log/storm/workers-artifacts/truck-event-processor-1-1474667899/6701 — ssh root@127.0.0.1 -p 2222 — 113×48
hbase(main):003:0> scan 'driver_dangerous_events' , {COLUMNS => ['events:hbaseRowKey:toString', 'events:latitude:
toDouble', 'events:longitude:toDouble', 'events:routeId:toInt', 'events:routeName:toString', 'events:truckId:toIn
t', 'events:driverId:toInt', 'events:driverName:toString', 'events:eventTime:toTimestamp', 'events:eventType:toSt
ring']}
ROW                          COLUMN+CELL
 10|58|922337056218683414    column=events:driverId, timestamp=1474668557140, value=10
 5
 10|58|922337056218683414    column=events:driverName, timestamp=1474668557140, value=George Vetticad
 5                           en
 10|58|922337056218683414    column=events:eventType, timestamp=1474668557140, value=Unsafe following
 5                            distance
 10|58|922337056218683414    column=events:hbaseRowKey, timestamp=1474668557140, value=10|58|92233705
 5                           62186834145
 10|58|922337056218683414    column=events:latitude, timestamp=1474668557140, value=37.02
 5
 10|58|922337056218683414    column=events:longitude, timestamp=1474668557140, value=-94.54
 5
 10|58|922337056218683414    column=events:routeId, timestamp=1474668557140, value=1390372503
 5
 10|58|922337056218683414    column=events:routeName, timestamp=1474668557140, value=Saint Louis to T
 5                           ulsa
 10|58|922337056218683414    column=events:truckId, timestamp=1474668557140, value=58
 5
 10|58|922337056218685984    column=events:driverId, timestamp=1474668557140, value=10
 3
 10|58|922337056218685984    column=events:driverName, timestamp=1474668557140, value=George Vetticad
 3                           en
 10|58|922337056218685984    column=events:eventType, timestamp=1474668557140, value=Lane Departure
 3
 10|58|922337056218685984    column=events:hbaseRowKey, timestamp=1474668557140, value=10|58|92233705
 3                           62186859843
 10|58|922337056218685984    column=events:latitude, timestamp=1474668557140, value=37.94
 3
 10|58|922337056218685984    column=events:longitude, timestamp=1474668557140, value=-91.99
 3
 10|58|922337056218685984    column=events:routeId, timestamp=1474668557140, value=1390372503
 3
 10|58|922337056218685984    column=events:routeName, timestamp=1474668557140, value=Saint Louis to T
 3                           ulsa
 10|58|922337056218685984    column=events:truckId, timestamp=1474668557140, value=58
 3
 10|58|922337056218688633    column=events:driverId, timestamp=1474668557140, value=10
 5
 10|58|922337056218688633    column=events:driverName, timestamp=1474668557140, value=George Vetticad
 5                           en
 10|58|922337056218688633    column=events:eventType, timestamp=1474668557140, value=Lane Departure
 5
 10|58|922337056218688633    column=events:hbaseRowKey, timestamp=1474668557140, value=10|58|92233705
```

- Once done, stop the Storm topology

Open the terminal of your sandbox:, then we can deactivate/kill the Storm topology from the Storm View or shell.

```
storm kill TruckEventKafkaExperimTopology
```



CONCLUSION

Congratulations, you built your first Hortonworks DataFlow Application.
When NiFi, Kafka and Storm are combined, they create the Hortonworks DataFlow.

You have used the power of NiFi to ingest, route and land real-time streaming data. You learned to capture that data with Kafka and perform instant processing with Storm. A common challenge with many use cases, which is also observed in this exercise series is ingesting a live stream of random data, and filtering the junk data from the actual data we care about. Through these exercises, you learned to manipulate, persist and perform many other operations on random data.

We have a working application that shows us a visualization of driver behavior, normal and dangerous events per city. Can you brainstorm ways to further enhance this application?