Hortonworks DataFlow

Getting Started with Streaming Analytics

(June 9, 2017)

docs.cloudera.com

Hortonworks DataFlow: Getting Started with Streaming Analytics

Table of Contents

1. Building an End-to-End Stream Application

A good way to get started using Hortonworks DataFlow (HDF) with Streaming Analytics Manager and Schema Registry is to imagine a real life use case, and to learn about the common HDF stream processing tasks and concepts through this use case. This guide sets up a fictional use case, and walks you through the deployment and common tasks you would perform while engaging in many of HDF's stream processing use cases.

Use this guide as a tutorial to get you started with SAM and Schema Registry. All the resources required to complete the tasks are provided in line.

1.1. Understanding the Use Case

To build a complex streaming analytics application from scratch, we will work with a fictional use case. A trucking company has a large fleet of trucks, and wants to perform real-time analytics on the sensor data from the trucks, and to monitor them in real time. Their analyitcs application has the following requirements:

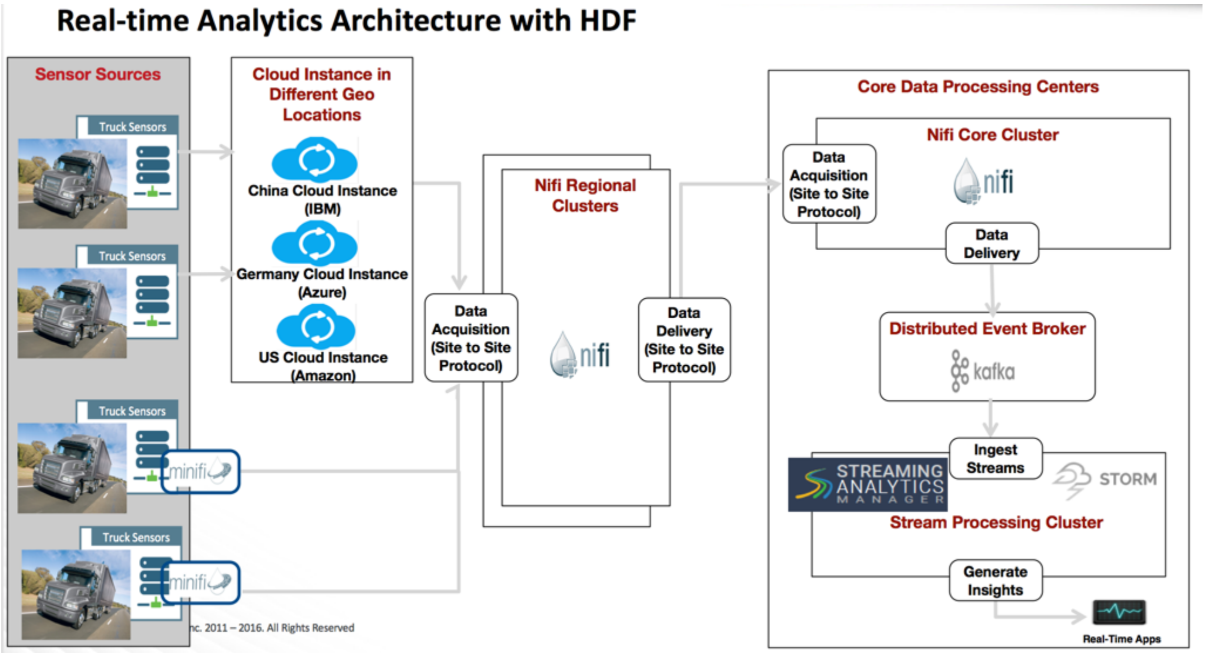
1. Outfit each truck with two sensors that emit event data such as timestamp, driver ID, truck ID, route, geographic location, and event type.
    - The geo event sensor emits geographic information (latitude and longitude coordinates) and events such as excessive braking or speeding.
    - The speed sensor emits the speed of the vehicle.
2. Stream the sensor events to an IoT gateway, which serializes the events as Avro objects and streams them into separate Kafka topics, one for each Kafka sensor.
3. Use NiFi to consume the serialized Avro events from the Kafka topics, and then route, transform, enrich, and deliver the data to a downstream Kafka instance.
4. Connect to the two streams of data to do analytics on the stream.
5. Join the two sensor streams using attributes in real-time. For example, join the geo-location stream of a truck with the speed stream of a driver.
6. Filter the stream on only events that are infractions or violations.
7. All infraction events need to be available for descriptive analytics (dash-boarding, visualizations, or similar) by a business analyst. The analyst needs the ability to do analysis on the streaming data.
8. Detect complex patterns in real-time. For example, over a three-minute period, detect if the average speed of a driver is more than 80 miles per hour on routes known to be dangerous.
9. When each of the preceding rules fires, create alerts and make them instantly accessible.

10. Execute a logistical regression Spark ML model on the events in the stream to predict if a driver is going to commit a violation. If violation is predicted, then alert on it.

The below sections walks you through how to implement all ten requirements. Requirements 1-3 are done using NiFi and Schema Registry. Requirements 4 through 10, are implemented using the new Streaming Analtyics Manager.

1.2. Reference Architecture

This reference architecture diagram gives you a general idea of how to build an HDF cluster for your trucking use case. Review this suggested architecture before you begin deployment.


Real-time Analytics Architecture with HDF

Sensor Sources
Truck Sensors
Truck Sensors
Truck Sensors
minifi
Truck Sensors
minifi
Cloud Instance in Different Geo Locations
China Cloud Instance (IBM)
Germany Cloud Instance (Azure)
US Cloud Instance (Amazon)
Data Acquisition (Site to Site Protocol)
Nifi Regional Clusters
nifi
Data Delivery (Site to Site Protocol)
Core Data Processing Centers
Data Acquisition (Site to Site Protocol)
Nifi Core Cluster
nifi
Data Delivery
Distributed Event Broker
kafka
Ingest Streams
STREAMING ANALYTICS MANAGER
STORM
Stream Processing Cluster
Generate Insights
Real-Time Apps
nc. 2011 – 2016. All Rights Reserved

2. Prepare Your Environment

2.1. Deploying Your HDF Clusters

About This Task

Now that you have reviewed the reference architecture and planned the deployment of your trucking application, you can begin installing HDF according to your use case specifications. To fully build the trucking application as described in this *Getting Started with Stream Analytics* document, use the following steps.

Steps

1. Install Ambari 2.5.1.

2. Install HDP 2.6.1 Cluster via Ambari.

3. Install HDF 3.0 Management Pack.

4. Update HDF 3.0 Base URL.

5. Add HDF 3.0 Services to HDP 2.6.1 cluster.

Find instructions for these installation steps in Installing HDF Services on a New HDP Cluster.

More Information

Planning Your Deployment

2.2. Registering Schemas in Schema Registry

The trucking application streams raw events that are serialized into Avro from the two sensors to its respective Kafka topics. NiFi consumes from these topics, and then routes, enriches, and delivers them to another set of Kafka topics for consumption by the streaming anlatyics applications. To do this, you must perform the following tasks:

- Creating the 4 Kafka topics

- Registering Schemas for each of the Kafka topics in the Schema Registry

2.2.1. Create the Kafka Topics

About This Task

Kafka topics are categories or feed names to which records are published.

Steps

1. Log into the node where Kafka broker is running.

2. Create the Kafka topics using the following commands:

```
cd /usr/[hdf/\hdp]current/kafka-broker/bin/

./kafka-topics.sh \
--create \
--zookeeper <zookeeper-host>:2181 \
--replication-factor 2 \
--partition 3 \
--topic raw-truck_events_avro

./kafka-topics.sh \
--create \
--zookeeper <zookeeper-host>:2181 \
--replication-factor 2 \
--partition 3 \
--topic raw-truck_speed_events_avro

./kafka-topics.sh \
--create \
--zookeeper <zookeeper-host>:2181 \
--replication-factor 2 \
--partition 3 \
--topic truck_events_avro


./kafka-topics.sh \
--create \
--zookeeper <zookeeper-host>:2181 \
--replication-factor 2 \
--partition 3 \
--topic truck_speed_events_avro
```

More Information

Apache Kafka Component Guide

2.2.2. Register Schemas for the Kafka Topics

About This Task

Register the schemas for the 2 Kafka topics that NiFi will consume from and the two other Kafka topics that NiFi will publish the enriched events to. Registering the Kafka topic schemas is benefiicial in several ways. Schema Registry provides a centralized schema location, allowing you to stream records into topics without having to attach the schema to each record.

Steps

1. Go to the Schema Registry UI by selecting the Registry service in Ambari and under 'Quick Links' selecting 'Registry UI'

2. Click the "+" button to add a schema, schema group and schema metadata for the Raw Geo Event Sensor Kafka topic:

   - `Name = raw-truck_events_avro`

   - `Description = Raw Geo events from trucks in Kafka Topic`

- `Type = Avro schema provider`
- `Schema Group = truck-sensors-kafka`
- `Compatibility: BACKWARD`
- Check the evolve check box
- Copy the schema from here and paste it into the Schema Text area.
- Click Save

3. Click the "+" button to add a schema, schema group (exists from previous step), and schema metadata for the Raw Speed Event Sensor Kafka topic:

    - `Name = raw-truck_speed_events_avro`
    - `Description = Raw Speed Events from trucks in Kafka Topic`
    - `Type = Avro schema provider`
    - `Schema Group = truck-sensors-kafka`
    - `Compatibility: BACKWARD`
    - Check the evolve check box
    - Copy the schema from here and paste it into the Schema Text area.
    - Click Save

4. Click the "+" button to add a schema, schema group and schema metadata for the Geo Event Sensor Kafka topic:

    - `Name = truck_events_avro`
    - `Description = Schema for the Kafka topic named 'truck_events_avro'`
    - `Type = Avro schema provider`
    - `Schema Group = truck-sensors-kafka`
    - `Compatibility: BACKWARD`
    - Check the evolve checkbox
    - Copy the schema from here and paste it into the Schema Text area.
    - Click Save

5. Click the "+" button to add a schema, schema group (exists from previous step), and schema metadata for the Speed Event Sensor Kafka topic:

    - `Name = truck_speed_events_avro`

- `Description = Schema for the Kafka topic named 'truck_speed_events_avro'`
- `Type = Avro schema provider`
- `Schema Group = truck-sensors-kafka`
- `Compatibility: BACKWARD`
- Check the evolve check box
- Copy the schema from here and paste it into the Schema Text area.
- Click **Save**.

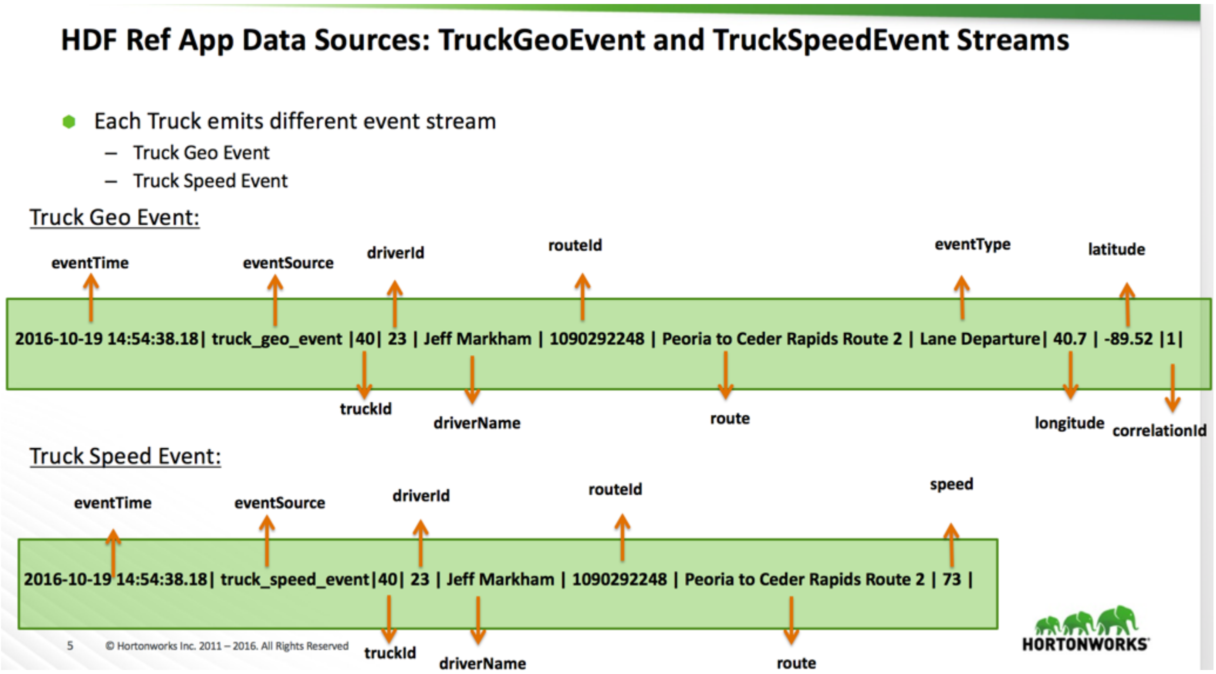
More Information

If you want to create these schemas programmatically using the Schema Registry client via REST rather than through the UI, you can find examples at this Github location.

3. Creating a Dataflow Application

3.1. Data Producer Application Generates Events

The following is a sample of a raw truck event stream generated by the sensors.



HDF Ref App Data Sources: TruckGeoEvent and TruckSpeedEvent Streams

- Each Truck emits different event stream
  - Truck Geo Event
  - Truck Speed Event

Truck Geo Event:

eventTime eventSource driverId routeId eventType latitude

2016-10-19 14:54:38.18| truck_geo_event |40| 23 | Jeff Markham | 1090292248 | Peoria to Ceder Rapids Route 2 | Lane Departure| 40.7 | -89.52 |1|

truckId driverName route longitude correlationId

Truck Speed Event:

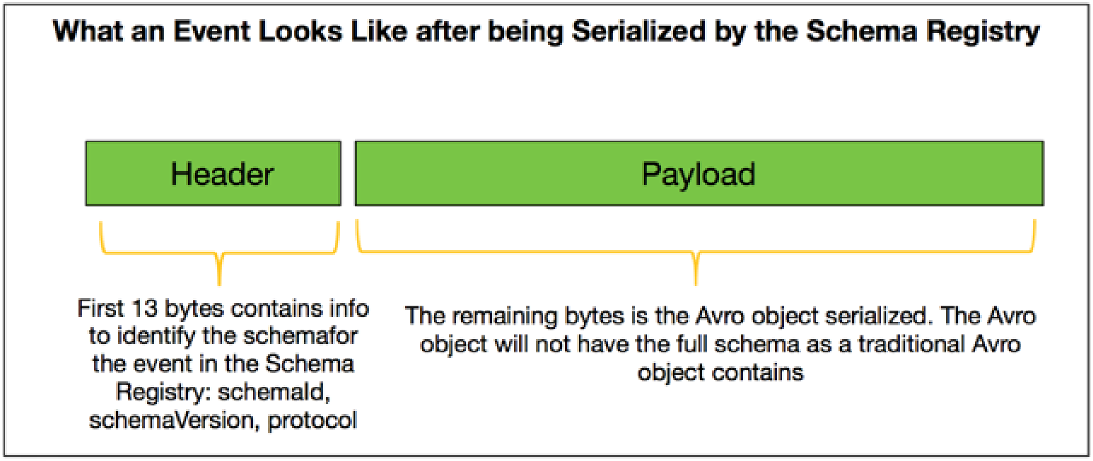
eventTime eventSource driverId routeId speed

2016-10-19 14:54:38.18| truck_speed_event|40| 23 | Jeff Markham | 1090292248 | Peoria to Ceder Rapids Route 2 | 73 |

truckId driverName route

5 © Hortonworks Inc. 2011 – 2016. All Rights Reserved

The date producing application or data simulator publishes these serialized Avro raw events into Kafka topics. The following is what the raw event looks like serialized into Avro using the Schema Registry.



What an Event Looks Like after being Serialized by the Schema Registry

Header

Payload

First 13 bytes contains info to identify the schemafor the event in the Schema Registry: schemaId, schemaVersion, protocol

The remaining bytes is the Avro object serialized. The Avro object will not have the full schema as a traditional Avro object contains

3.2. NiFi: Create a Dataflow Application

To make things easier to setup, import the NiFi Template for this flow by downloading it this Github location. After importing, select Use Case 1 process group. The below instructions are with respect to that flow.

3.2.1. NiFi Controller Services

Click on Flow Configuration Settings icon and select Controller Services tab.

Hortonworks Schema Registry Controller Service

1. Click on Flow Configuration Settings icon and select **Controller Services** tab.

2. You will see the HWX Schema Registry controller service. Edit the properties to configure the Schema Registry URL based on your environment. You can find this value in the Streaming Analytics Manager Service in Ambari for the configuration property called `registry.url`. An example of what the URL looks similar to `http://$REGISTRY_SERVER:7788/api/v1`.

3. Enable this controller service.

RecordReader and RecordWriter Controller Services

The RecordReader and RecordWriter controller services are new controller services that allows you convert events from one type (json, xml, csv, Avro) to another (json, xml, csv, Avro). These controller services use the Schema Registry to fetch the schema for the event to do this conversion. There are a number of different schema access strategies you can configure on the RecordReader and RecordWriter to tell the Record Reader/Writer how to look up the schema information. For example, if you are reading records serialized by the Hortonworks Schema Registry, the schema identifier required to look up the schema in the registry is embedded in the header of the payload. Hence, the RecordReader would use the schema access strategy called "HWX Content-Encoded Schema Reference". The following are the RecordReader and RecordWriter controller services used for the NiFi template imported:

- Avro Truck Events - Reads Avro events and looks up the schema id via the HWX Content-Encoded Schema Reference strategy. This schema id is then used to query the schema from the Hortonworks Schema Registry.

- CSV Truck Events - Reads csv events and looks up the schema name from the value of the "Schema Name" attribute. This schema lookup strategy is called the "Use 'Schema Name' Property" access strategy. This value of this schema name property is then used to query the schema from the Hortonworks Schema Registry.

- AvroRecordSetWriter - Writes events into Avro and looks up the schema identifier info using the HWX Schema Reference Attribute strategy. This controller also uses a write strategy of HWX Content-Encoded Schema Reference where the Avro object will have schema identifier information pre-appended on the header.

- AvroRecordSetWriter-Read-Schema-From-HWX-Via-Schema-Name - Writes events into Avro and looks up the schema using the Schema Name access strategy. This controller also uses a write strategy of HWX Content-Encoded Schema Reference where the Avro object will have schema identifier information pre-appended on the header.

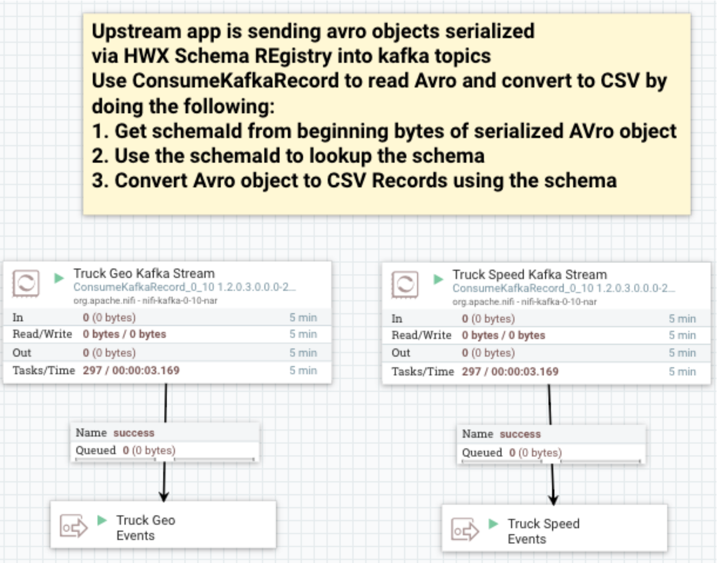
- CSVRecordSetWriter - Writes events into CSV and looks up the schema identifier using the HWX Schema Reference Attribute strategy. The write schema strategy is also HWX Schema Reference attributes. This means when the csv is written the schema identifier information is stored in named attributes of the flow file.

- CSVRecordSetWriter-Read-Schema-From-HWX-Embedded - Similar to the previous csv writer but the schema identifier is looked using the HWX Content-Encoded Schema Reference strategy.

Enable all of these controller services.

3.2.2. NiFi Ingests the Raw Sensor Events

In the Use Case 1 process group, go into the "Acquire Events" process group. The first step in the NiFi flow is to ingest the raw serialized Avro events from the two Kafka topics. We will use the new ConsumerKafkaRecord processor for this.



Upstream app is sending avro objects serialized via HWX Schema REgistry into kafka topics
Use ConsumeKafkaRecord to read Avro and convert to CSV by doing the following:
1. Get schemaId from beginning bytes of serialized AVro object
2. Use the schemaId to lookup the schema
3. Convert Avro object to CSV Records using the schema

Truck Geo Kafka Stream
ConsumeKafkaRecord_0_10 1.2.0.3.0.0.0-2...
org.apache.nifi - nifi-kafka-0-10-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	297 / 00:00:03.169	5 min

Name success
Queued 0 (0 bytes)

Truck Geo Events

Truck Speed Kafka Stream
ConsumeKafkaRecord_0_10 1.2.0.3.0.0.0-2...
org.apache.nifi - nifi-kafka-0-10-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	297 / 00:00:03.169	5 min

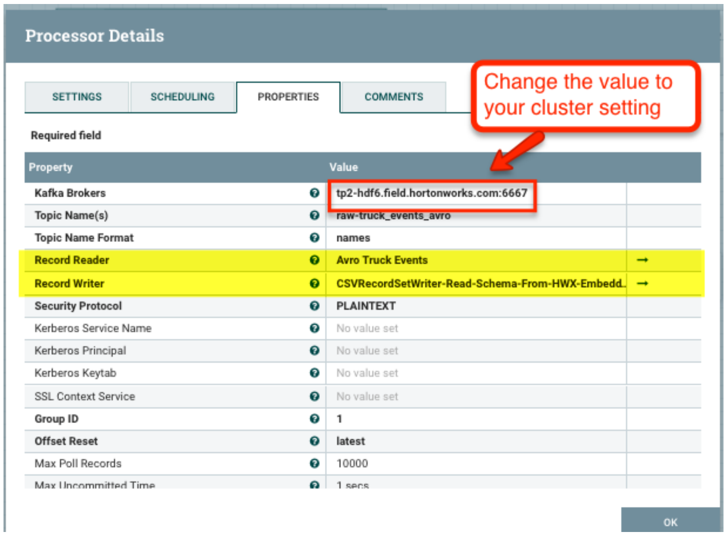
Name success
Queued 0 (0 bytes)

Truck Speed Events

Both ConsumerKafkaRecord processors are configured with an AvroReader controller service and the CSVRecordSetWriter-Read-Schema-From-HWX-Embedded controller service to convert from Avro to CSV using a schema.

Note

Make sure for both processors, you change the Kafka Brokers property value to your cluster settings.



Processor Details

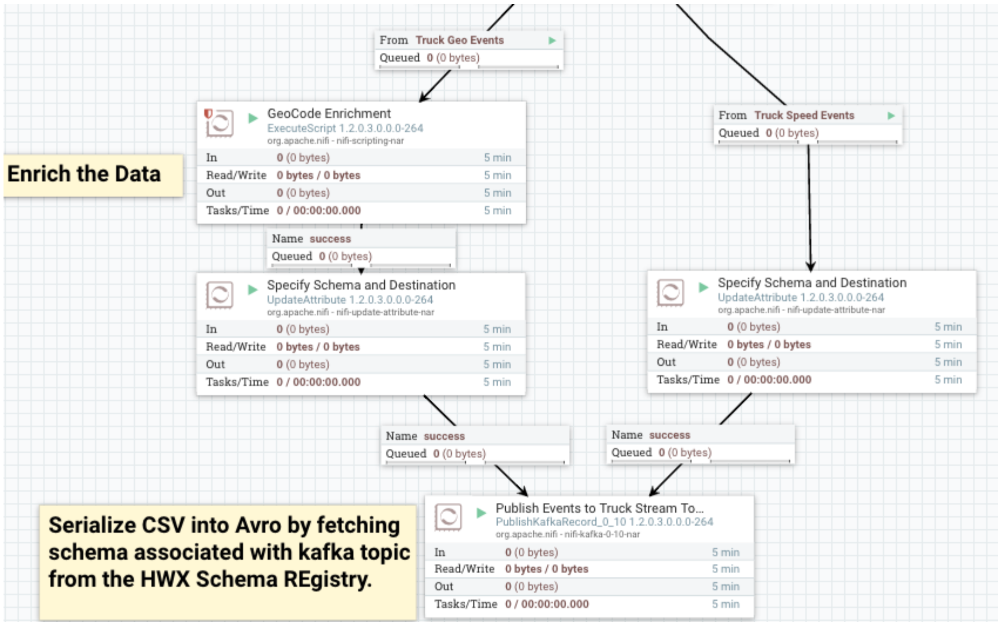
Change the value to your cluster setting

Required field

Property	Value
Kafka Brokers	tp2-hdf6.field.hortonworks.com:6667
Topic Name(s)	raw-truck_events_avro
Topic Name Format	names
Record Reader	Avro Truck Events
Record Writer	CSVRecordSetWriter-Read-Schema-From-HWX-Embedd...
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Group ID	1
Offset Reset	latest
Max Poll Records	10000
Max Uncommitted Time	1 secs

3.2.3. Publish Enriched Events to Kafka for Consumption by Analytics Applications

After NiFi has done the routing, transforms, and enrichment, NiFi will publish the enriched events into Kafka topics. These topics have a schema registered for it in the Schema Registry and we will store the schema identifier for the schema in the FlowFile attributes (UpdateAttribute processors) and use the PublishKafkaRecord processor to push the events into Kafka.



Enrich the Data

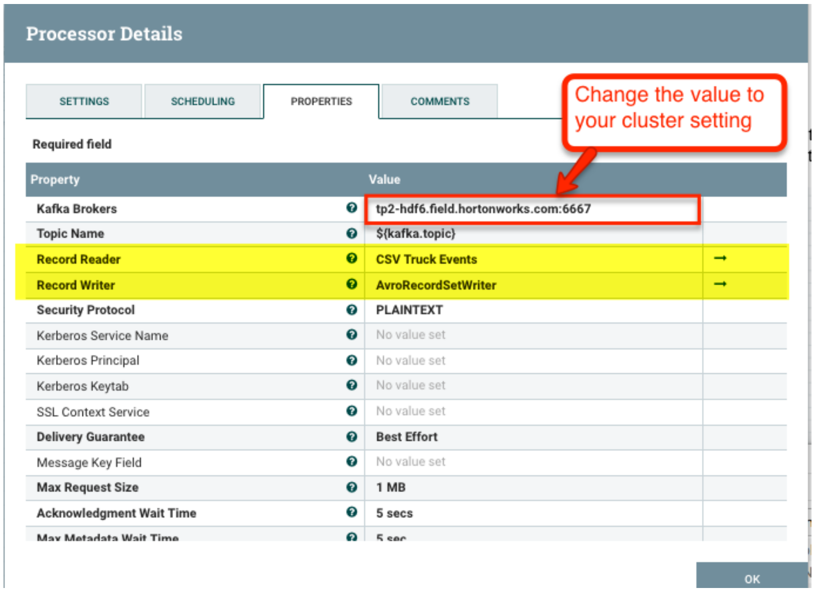
Serialize CSV into Avro by fetching schema associated with kafka topic from the HWX Schema REgistry.

From Truck Geo Events
Queued 0 (0 bytes)
GeoCode Enrichment
ExecuteScript 1.2.0.3.0.0.0-264
org.apache.nifi - nifi-scripting-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Specify Schema and Destination
UpdateAttribute 1.2.0.3.0.0.0-264
org.apache.nifi - nifi-update-attribute-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
From Truck Speed Events
Queued 0 (0 bytes)
Specify Schema and Destination
UpdateAttribute 1.2.0.3.0.0.0-264
org.apache.nifi - nifi-update-attribute-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
Publish Events to Truck Stream To...
PublishKafkaRecord_0_10 1.2.0.3.0.0.0-264
org.apache.nifi - nifi-kafka-0-10-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

The PublishKafkaRecord processor is configured with the controller service 'CSV Truck Events' for the Record Reader and uses the AvroRecordSetWriter to write the events into Avro. It is a serialized Avro object with the schema identifier in the header that gets published to Kafka for consumption by SAM.

Note

Make sure for the PublishKafkaRecord, you change the Kafka Brokers property value to your cluster settings.



Change the value to your cluster setting

Required field

Property	Value
Kafka Brokers	tp2-hdf6.field.hortonworks.com:6667
Topic Name	${kafka.topic}
Record Reader	CSV Truck Events
Record Writer	AvroRecordSetWriter
Security Protocol	PLAINTEXT
Kerberos Service Name	No value set
Kerberos Principal	No value set
Kerberos Keytab	No value set
SSL Context Service	No value set
Delivery Guarantee	Best Effort
Message Key Field	No value set
Max Request Size	1 MB
Acknowledgment Wait Time	5 secs
Max Metadata Wait Time	5 sec

3.2.4. Start the NiFi Flow

Start the Process Grouped called "Use Case 1".

4. Creating a Stream Analytics Application

4.1. Create a Service Pool and Environment

Before you create an application, you have to create a Service Pool and then an Environment that you associate with an application. Refer to the *Streaming Analytics Manager User Guide* sections on Streaming Analytics Manager Environment Setup and Managing Stream Applications.

4.2. Create Your First Application

About This Task

The Streaming Analtyics Manager provides capabilities to the application developer for building streaming applications. You can go to the Stream Builder UI by select the **Streaming Analytics Manager** service in Ambari and under **Quick Links** select **SAM UI**.

Creating a new stream application requires two steps: clicking the + icon, and then providing a unique name for the stream application and associating the application with an Environment.

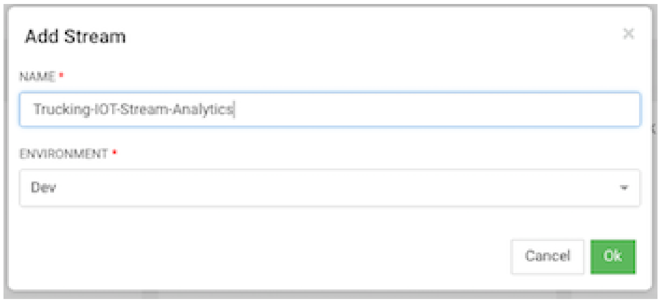
Steps

1. Click the + icon on the **My Applications** dashboard and choose **New Application**.

2. Specify the name of the stream application and the environment that you want it to use stream.

Note

The name of the stream application cannot have any spaces.



Add Stream

NAME *

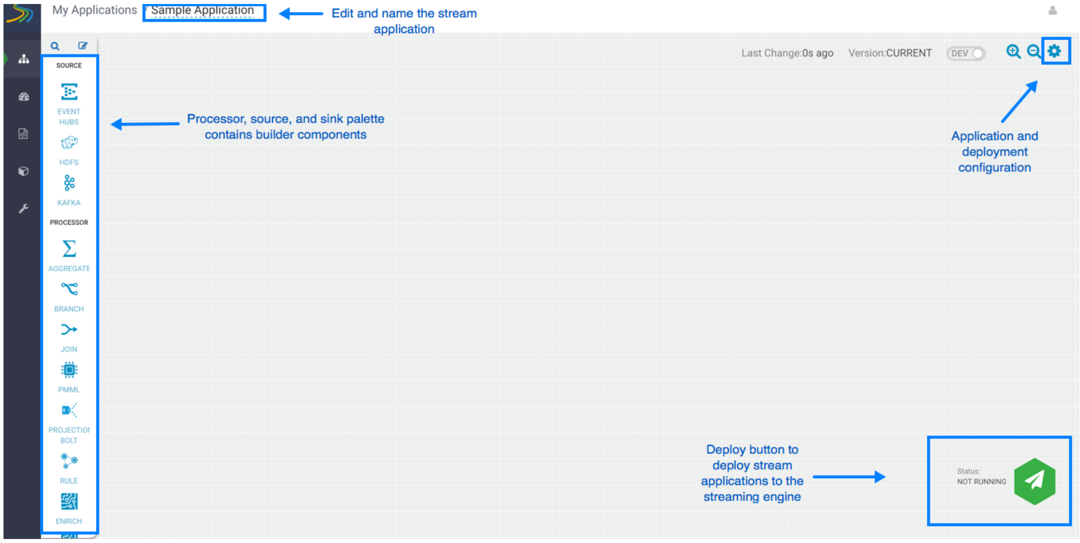
Trucking-IOT-Stream-Analytics

ENVIRONMENT *

Dev

Result

SAM displays the Stream Builder canvas. Builder components on the canvas palette are the building blocks used to build stream applications. Now you are ready to start building the streaming application.



My Applications
Sample Application
Edit and name the stream application
Last Change:0s ago
Version:CURRENT
DEV
SOURCE
EVENT HUBS
HDFS
KAFKA
PROCESSOR
AGGREGATE
BRANCH
JOIN
PMML
PROJECTION BOLT
RULE
ENRICH
Processor, source, and sink palette contains builder components
Application and deployment configuration
Deploy button to deploy stream applications to the streaming engine
Status:
NOT RUNNING

4.3. Creating and Configuring the Kafka Source Stream

About This Task

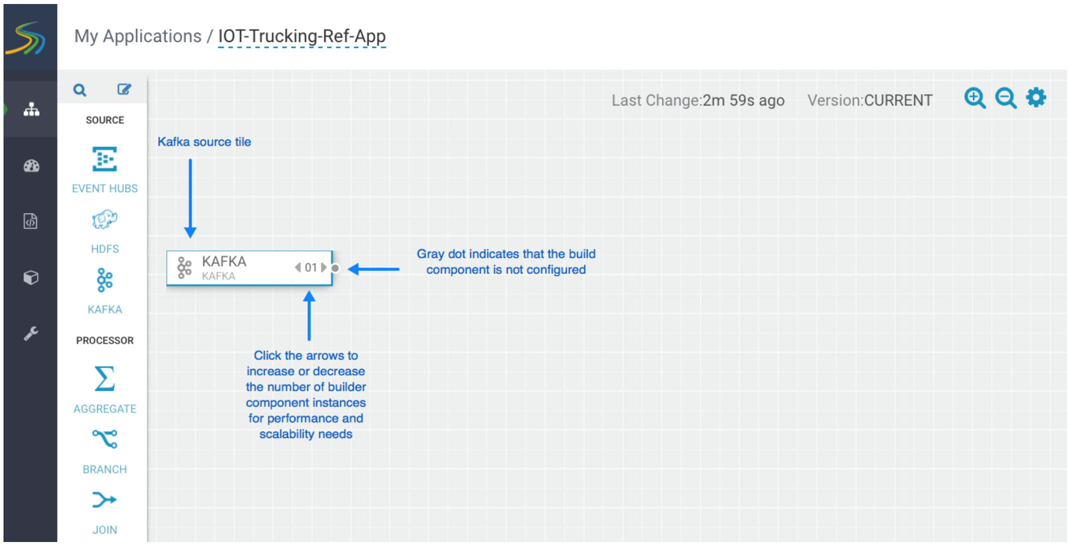
The first step in building a stream application is to drag builder components to the canvas. As described in the *Hortonworks DataFlow Overview*, Stream Builder offers four types of builder components: sources, processors, sinks, and custom components.

Every stream application must start with a source.

Complete the following instructions to start building a stream application. Use these steps to implement Requirement 4 of the use case.

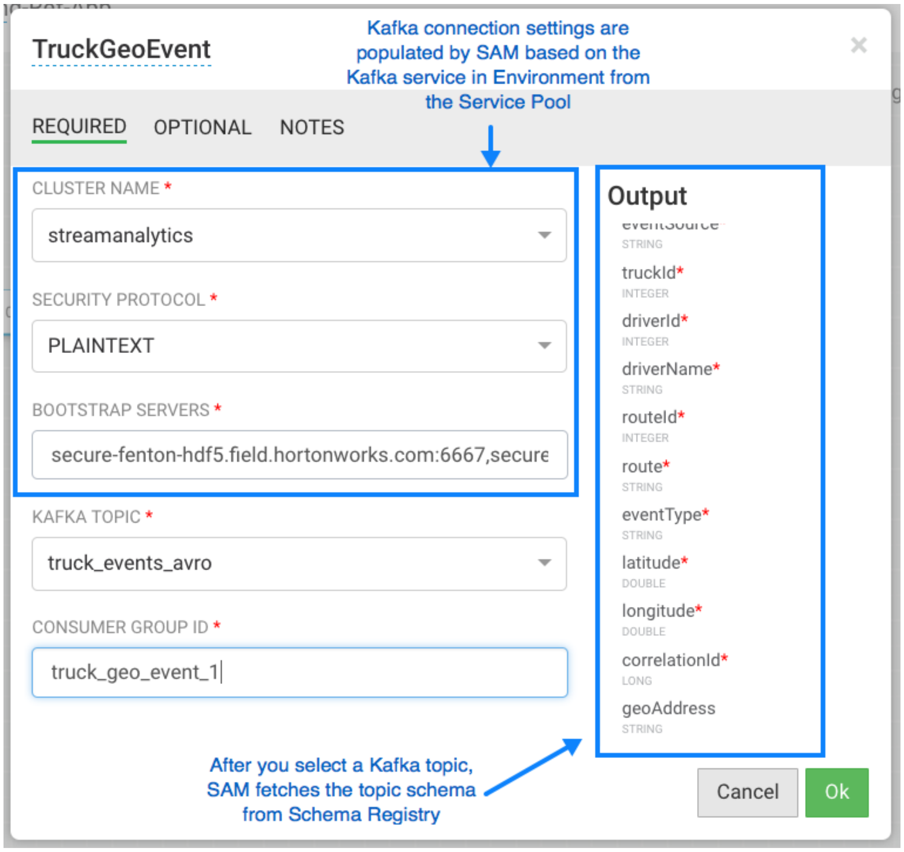
Steps

1. Drag the Kafka builder component onto the canvas, creating a Kafka tile:


My Applications / IOT-Trucking-Ref-App
Last Change:2m 59s ago
Version:CURRENT
SOURCE
EVENT HUBS
HDFS
KAFKA
PROCESSOR
AGGREGATE
BRANCH
JOIN
KAFKA
KAFKA
01
Kafka source tile
Gray dot indicates that the build component is not configured
Click the arrows to increase or decrease the number of builder component instances for performance and scalability needs

2. Set the number of run-time instances for your Kafka tile component by clicking the up arrow on the tile.

3. Double-click on the tile to begin configuring Kafka. After you specify a Kafka topic name, SAM communicates with the Schema Registry and displays the schema:



TruckGeoEvent

Kafka connection settings are populated by SAM based on the Kafka service in Environment from the Service Pool

CLUSTER NAME * streamanalytics

SECURITY PROTOCOL * PLAINTEXT

BOOTSTRAP SERVERS * secure-fenton-hdf5.field.hortonworks.com:6667,secure

KAFKA TOPIC * truck_events_avro

CONSUMER GROUP ID * truck_geo_event_1

Output

eventSource	STRING
truckId*	INTEGER
driverId*	INTEGER
driverName*	STRING
routeId*	INTEGER
route*	STRING
eventType*	STRING
latitude*	DOUBLE
longitude*	DOUBLE
correlationId*	LONG
geoAddress	STRING

After you select a Kafka topic, SAM fetches the topic schema from Schema Registry

Result

Once you have configured your Kafka component correctly, the tile component displays a green dot.

More Information

Hortonworks DataFlow Overview

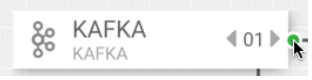
4.4. Connecting Components

About This Task

To pass a stream of events from one component to the next, create a connection between the two components. In addition to defining data flow, connections allow you to pass a schema from one component to another.

Steps

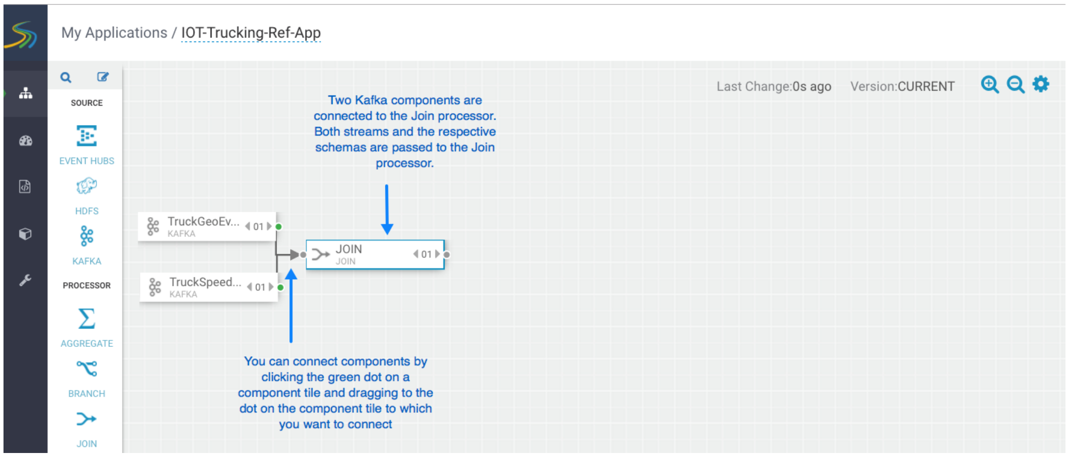
1. Click the green dot to the left of your source component.



2. Drag your cursor to the component tile to which you want to connect.

Example

The following example shows two connections: a connection from Kafka sink TruckGeoStream to the join processor, and a connection from the Kafka sink TruckSpeedStream to the same join processor.



My Applications / IOT-Trucking-Ref-App

Last Change:0s ago Version:CURRENT

Two Kafka components are connected to the Join processor. Both streams and the respective schemas are passed to the Join processor.

You can connect components by clicking the green dot on a component tile and dragging to the dot on the component tile to which you want to connect

4.5. Joining Multiple Streams

About This Task

Joining multiple streams is an important SAM capability. You accomplish this by adding the Join processor to your strream application.

This section shows you how to configure a Join processor that joins the truck geo-event stream with the speed event stream, based on Requirement 5 of the use case.

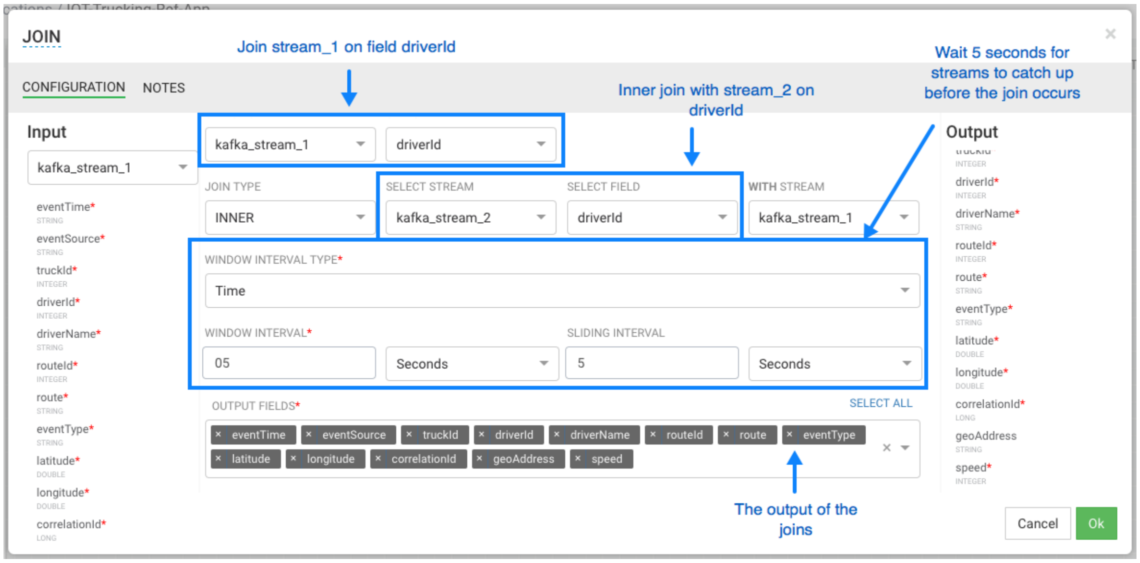
Steps

1. Drag a Join processor onto your canvas and connect it to a source.

2. Double click the Join tile to open the **Configuration** dialog.

3. Configure the Join processors according to the example below.

Example



JOIN

CONFIGURATION NOTES

Input: kafka_stream_1

Join stream_1 on field driverId

kafka_stream_1 | driverId

JOIN TYPE: INNER

Inner join with stream_2 on driverId

SELECT STREAM: kafka_stream_2 | SELECT FIELD: driverId | WITH STREAM: kafka_stream_1

Wait 5 seconds for streams to catch up before the join occurs

WINDOW INTERVAL TYPE*: Time

WINDOW INTERVAL*: 05 Seconds | SLIDING INTERVAL: 5 Seconds

OUTPUT FIELDS*: eventTime, eventSource, truckId, driverId, driverName, routeId, route, eventType, latitude, longitude, correlationId, geoAddress, speed

The output of the joins

4.6. Filtering Events in a Stream using Rules

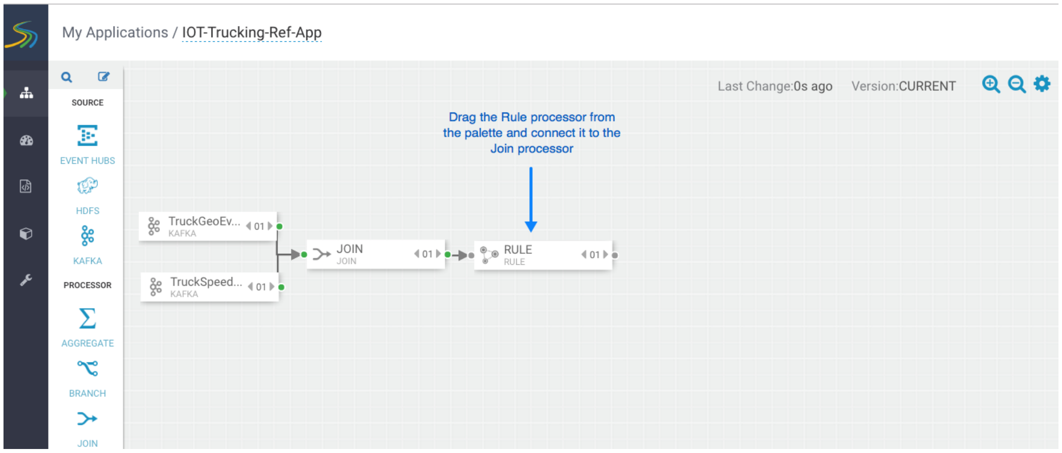
About This Task

SAM provides powerful capabilities to filter events in the stream. It uses a Rules Engine, which translates rules into SQL queries that operate on the stream of data.

The following steps demonstrate this, implementing Requirement 6 of the use case.

Steps

1. Drag the Rule processor to the canvas and connect it from the Join processors.

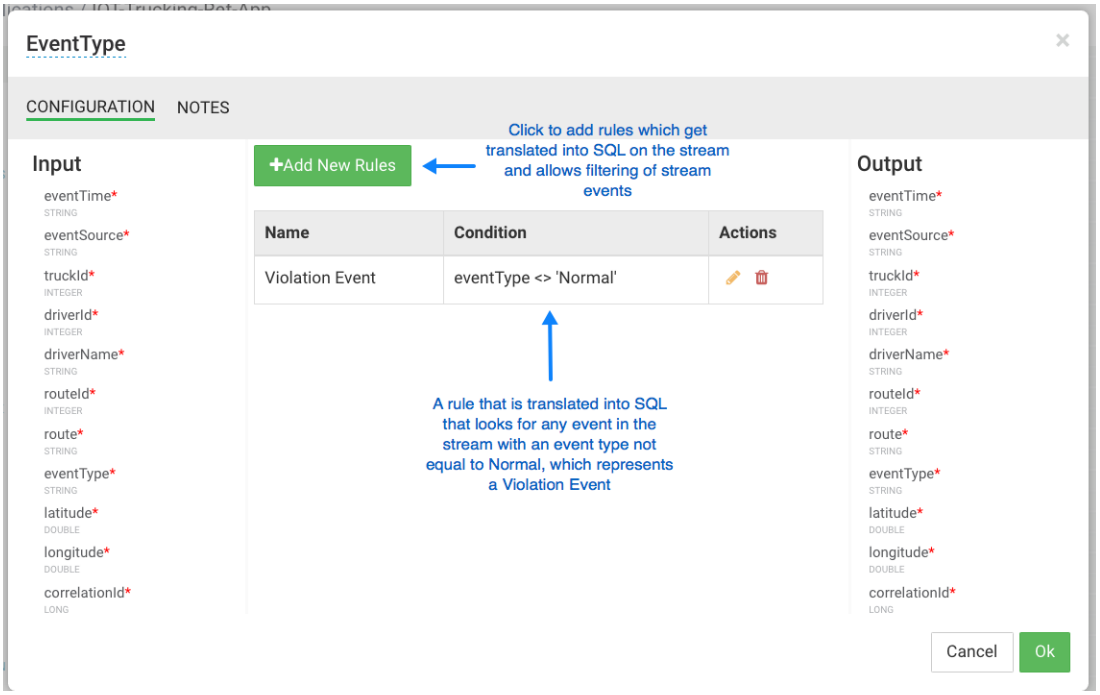


Drag the Rule processor from the palette and connect it to the Join processor

2. Double click the Rule processor, click the **Add new Rules** button, and create a new rule.

3. Click **OK** to save the new rule.

Example



EventType

CONFIGURATION NOTES

Click to add rules which get translated into SQL on the stream and allows filtering of stream events

Name	Condition	Actions
Violation Event	eventType <> 'Normal'	

A rule that is translated into SQL that looks for any event in the stream with an event type not equal to Normal, which represents a Violation Event

4.7. Using Aggregate Functions over Windows

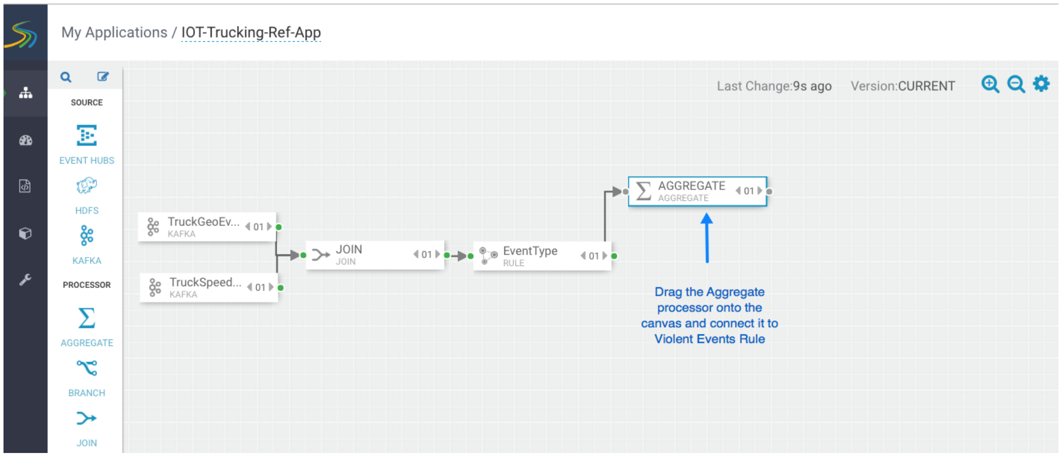
About This Task

Windowing is the ability to split an unbounded stream of data into finite sets based on specified criteria such as time or count, so that you can perform aggregate functions (such as sum or average) on the bounded set of events. SAM exposes these capabilities using the Aggregate processor. The Aggregate processor supports two window types, tumbling and sliding windows. The creation of a window can be based on time or count.

The following images show how to use the Aggregate processor to implement Requirement 8 of the use case.

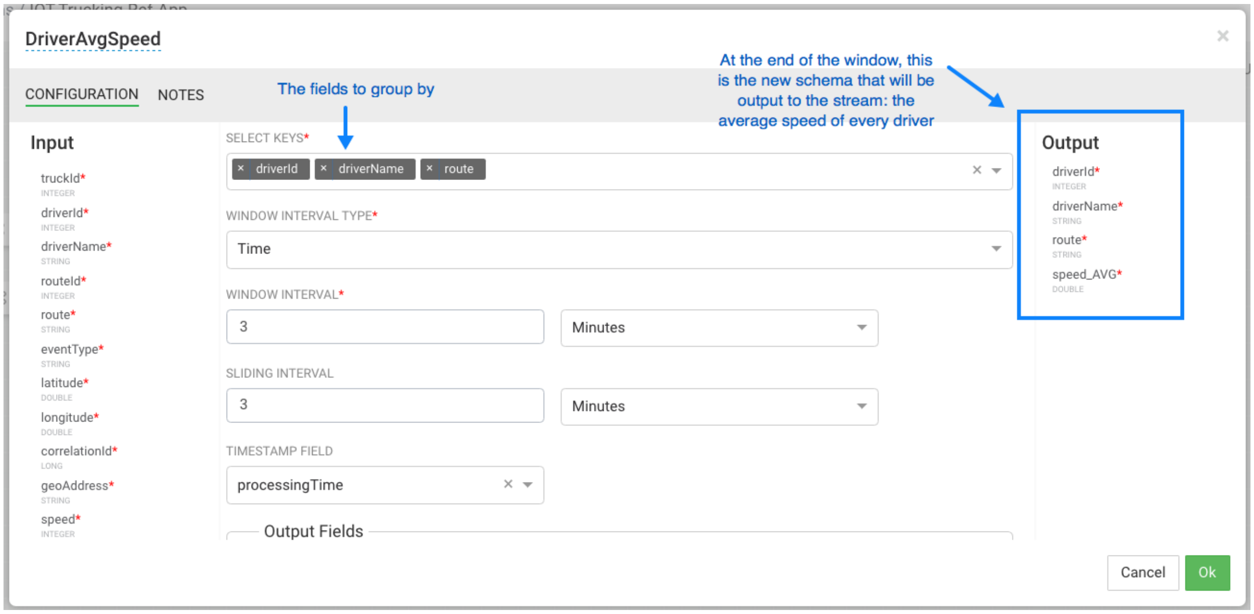
Steps

1. Drag the Aggregate processor to the canvas and connect it to the Rule processor.



Drag the Aggregate processor onto the canvas and connect it to Violent Events Rule

2. Double-click on the Aggregate processor, and configure it to calculate the average speed of driver over a three-minute duration.



DriverAvgSpeed

CONFIGURATION NOTES

The fields to group by

At the end of the window, this is the new schema that will be output to the stream: the average speed of every driver

Input

- truckId* INTEGER
- driverId* INTEGER
- driverName* STRING
- routeId* INTEGER
- route* STRING
- eventType* STRING
- latitude* DOUBLE
- longitude* DOUBLE
- correlationId* LONG
- geoAddress* STRING
- speed* INTEGER

SELECT KEYS*: driverId, driverName, route

WINDOW INTERVAL TYPE*: Time

WINDOW INTERVAL*: 3 Minutes

SLIDING INTERVAL: 3 Minutes

TIMESTAMP FIELD: processingTime

Output Fields

Output

- driverId* INTEGER
- driverName* STRING
- route* STRING
- speed_AVG* DOUBLE

4.8. Implementing Business Rules on the Stream

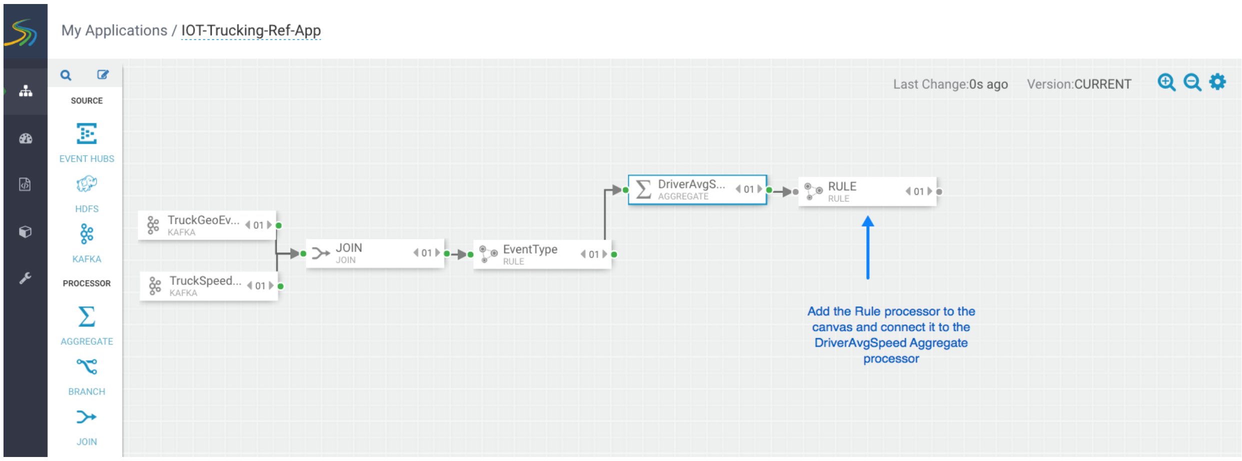
About This Task

This section shows you how to implement the business rule you created above to detect high speeding drivers. "High speed" is defined as greater than 80 miles per hour over a three-minute time window.

This step partially implements Requirement 8 of the use case.

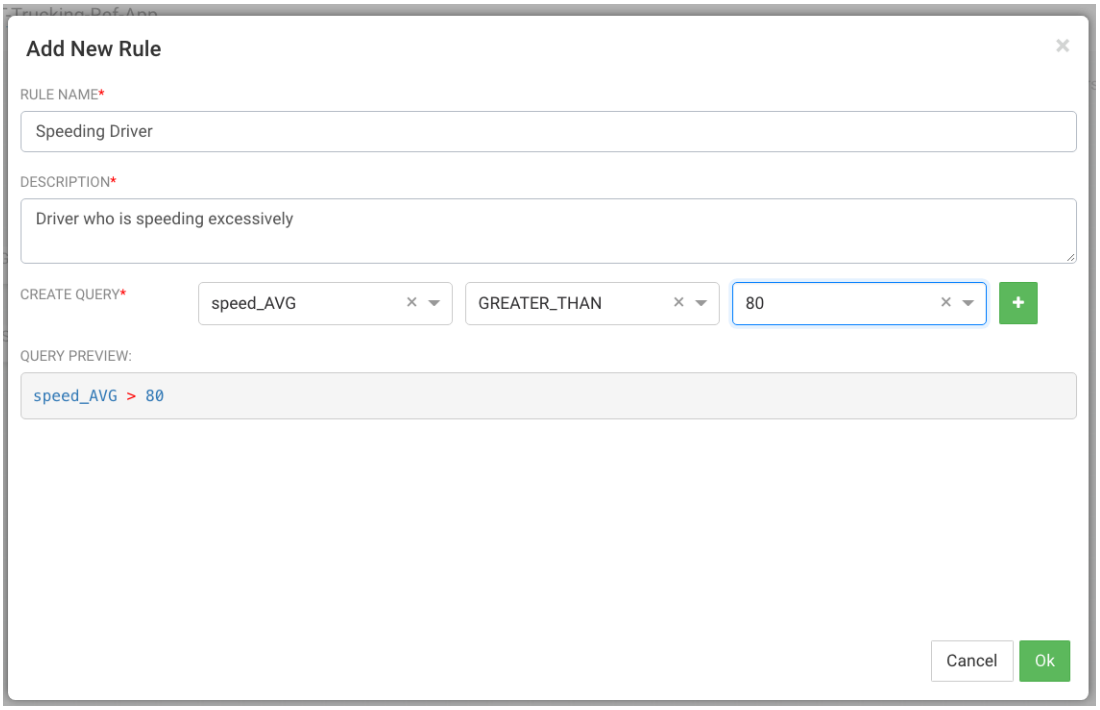
Steps

1. Drag the Rule processor onto the canvas and connect to it to the DriverAvgSpeed Aggregate processor:



Add the Rule processor to the canvas and connect it to the DriverAvgSpeed Aggregate processor

2. Configure the business rule as follows:



Add New Rule

RULE NAME*

Speeding Driver

DESCRIPTION*

Driver who is speeding excessively

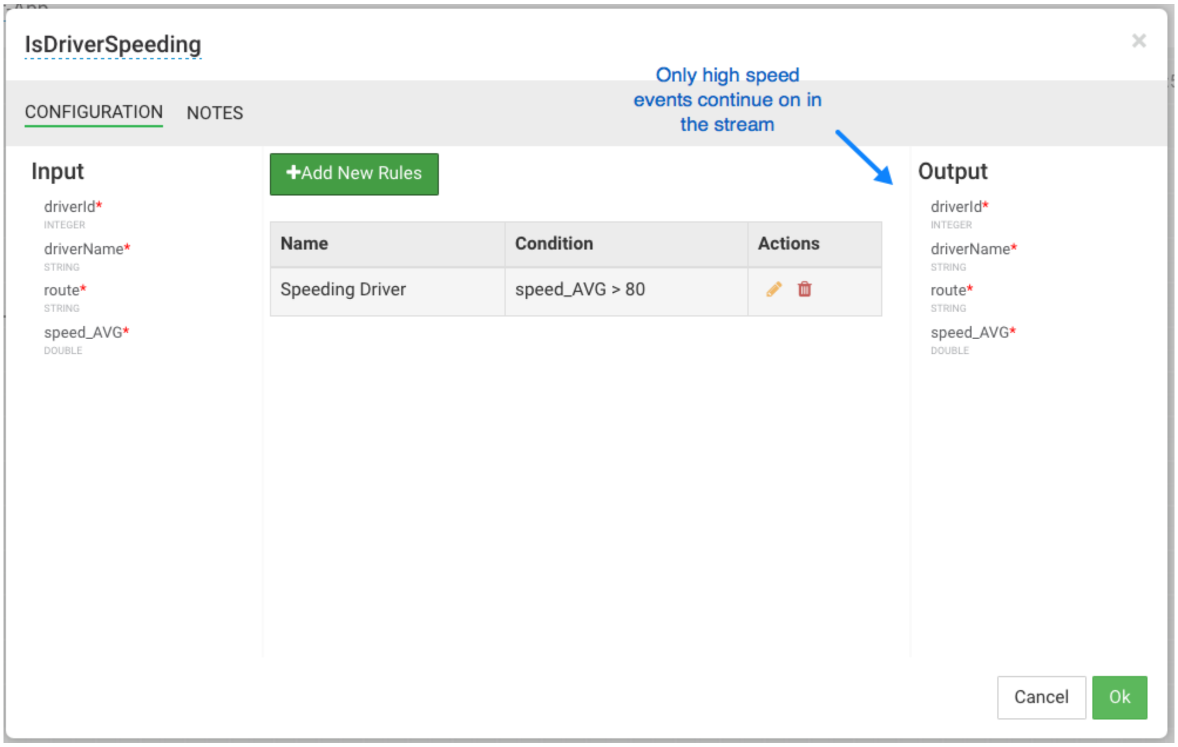
CREATE QUERY*: speed_AVG | GREATER_THAN | 80

QUERY PREVIEW:

```
speed_AVG > 80
```

Result

The fully configured business rule should look similar to the following. Only high speed events continue on in the stream.



IsDriverSpeeding

CONFIGURATION NOTES

Only high speed events continue on in the stream

Input

- driverId* INTEGER
- driverName* STRING
- route* STRING
- speed_AVG* DOUBLE

+Add New Rules

Name	Condition	Actions
Speeding Driver	speed_AVG > 80	✎ 🗑

Output

- driverId* INTEGER
- driverName* STRING
- route* STRING
- speed_AVG* DOUBLE

Cancel Ok

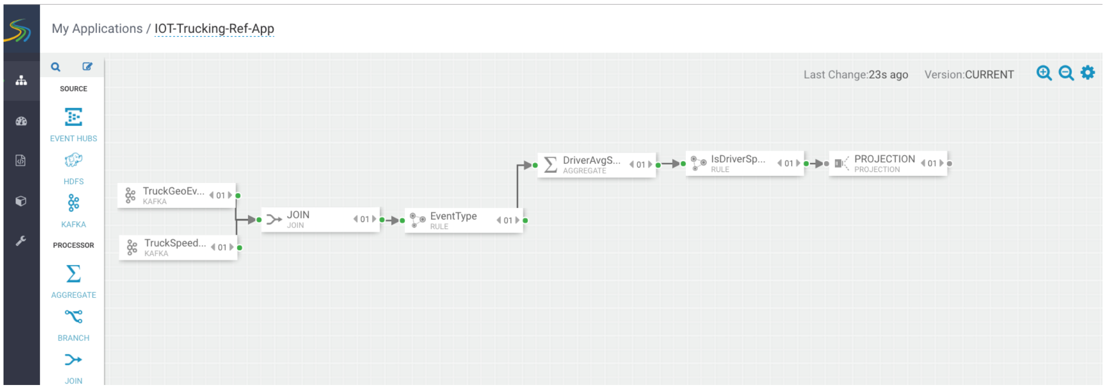
4.9. Transforming Data using a Projection Processor

About This Task

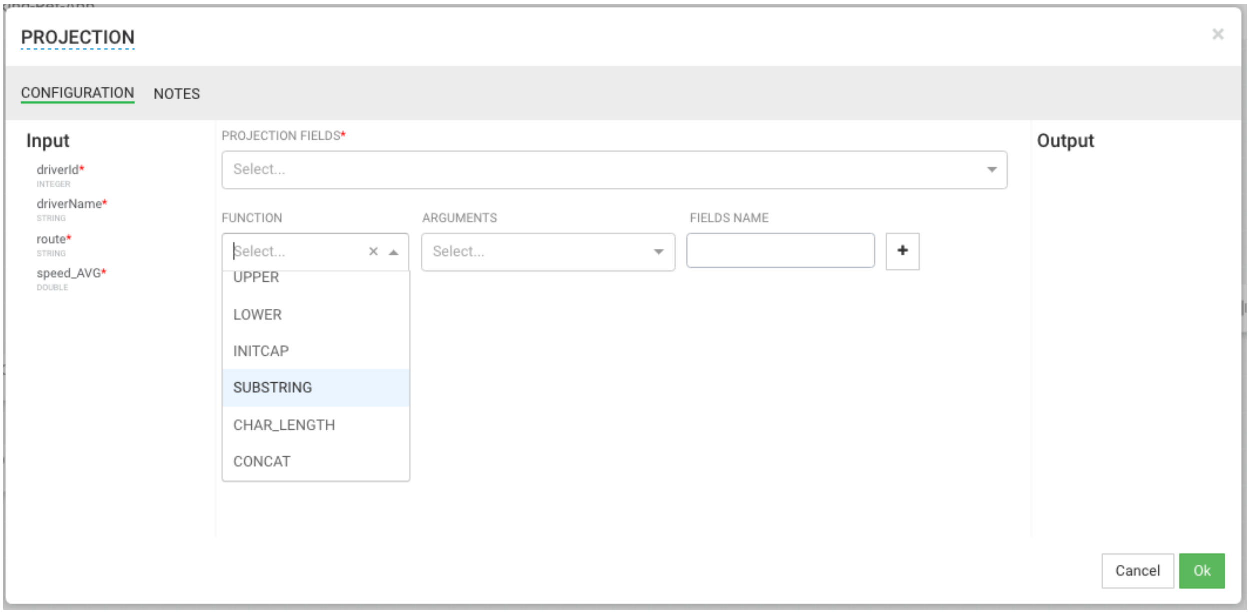
It is common to do transformations on the events in the stream. In our case, before we alert on the speeding driver, we want to convert the average speed we calculated in the aggregate processor into a integer from a double so it is easier to display in the alert. The projection processor allows you to do these transformations.

Steps

1. Drag the Projection processor onto the canvas and connect to it to the IsDriverSpeeding Rule processor:



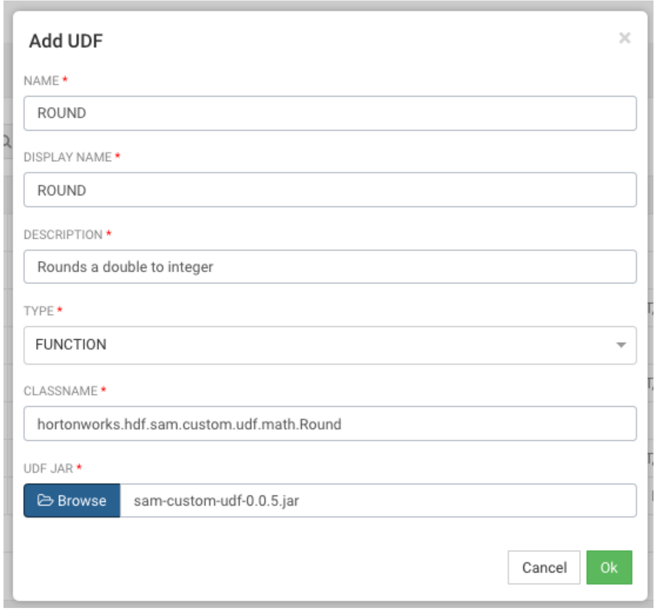
2. When you double click on the projection processor, you will see a number of out of the box functions however a Round function does not exist.



3. Adding UDFs (User Defined Functions) is easy to do within SAM. Follow the below steps to add Round UDF function to SAM.

   a. From the left-hand menu, click **Configuration**, then **Application Resources**.

   b. Select the **UDF** tab and click the + sign to create the ROUND UDF. The `jar` for this UDF can be downloaded from here. The simple java class used to implement this Round function using the SAM SDK can be found here. Unzip the downloaded artifact and use the jar called sam-custom-udf-0.0.5.jar. Configure the UDF with the following values:


Add UDF

NAME *: ROUND

DISPLAY NAME *: ROUND

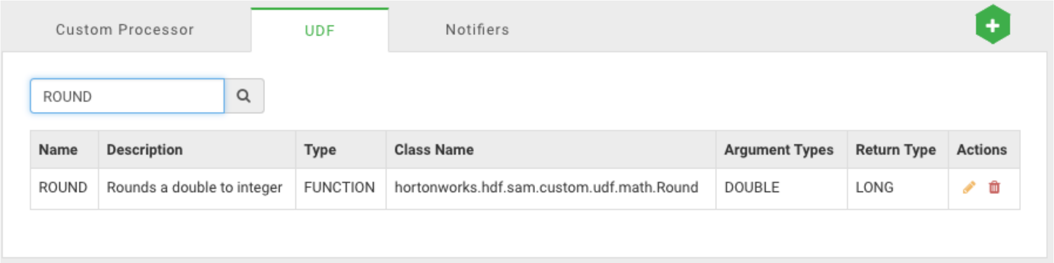
DESCRIPTION *: Rounds a double to integer

TYPE *: FUNCTION

CLASSNAME *: hortonworks.hdf.sam.custom.udf.math.Round

UDF JAR *: sam-custom-udf-0.0.5.jar

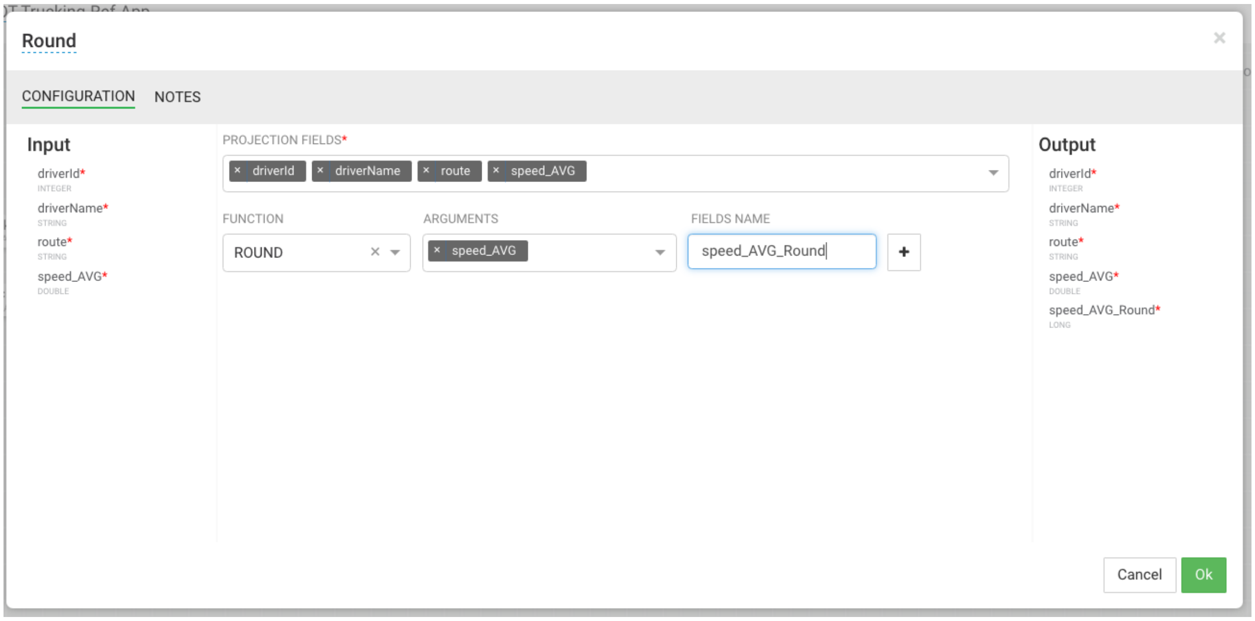
c. After uploading the UDF, you should see the new Round UDF created.



ROUND

Name	Description	Type	Class Name	Argument Types	Return Type	Actions
ROUND	Rounds a double to integer	FUNCTION	hortonworks.hdf.sam.custom.udf.math.Round	DOUBLE	LONG	✎ 🗑

4. After creating the UDF, go back to your Application and double click on the on the Projection Processor you added to the canvas and you see ROUND in the **FUNCTION** drop down list. Configure the ROUND function as the following:



Input: driverId* INTEGER, driverName* STRING, route* STRING, speed_AVG* DOUBLE

PROJECTION FIELDS*: driverId, driverName, route, speed_AVG

FUNCTION: ROUND

ARGUMENTS: speed_AVG

FIELDS NAME: speed_AVG_Round

Output: driverId* INTEGER, driverName* STRING, route* STRING, speed_AVG* DOUBLE, speed_AVG_Round* LONG

4.10. Creating Alerts with Notifications Sink

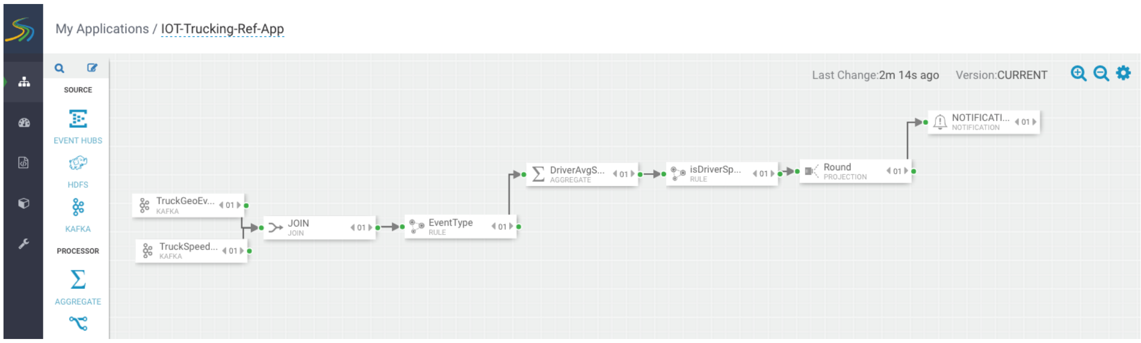
About This Task

The Notifications sink allows you to create alerts. The Notification sink supports email alerts, and it is extensible– you can plug in other types of notifications.

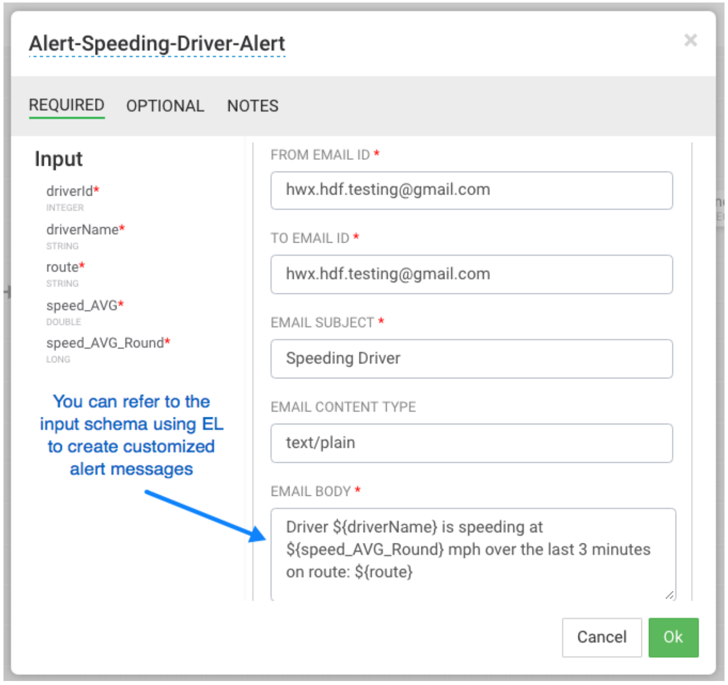
The following steps demonstrate how to create email alerts when drivers are speeding. Like custom UDFs, custom notifications can be added to SAM.

Steps

1. Drag the Notifications sink to the canvas and connect to it to the Round Projection.



2. Configure the Notifications sink to generate email alerts for high speeding drivers.



Alert-Speeding-Driver-Alert

REQUIRED OPTIONAL NOTES

Input

driverId* INTEGER

driverName* STRING

route* STRING

speed_AVG* DOUBLE

speed_AVG_Round* LONG

FROM EMAIL ID *: hwx.hdf.testing@gmail.com

TO EMAIL ID *: hwx.hdf.testing@gmail.com

EMAIL SUBJECT *: Speeding Driver

EMAIL CONTENT TYPE: text/plain

EMAIL BODY *: Driver ${driverName} is speeding at ${speed_AVG_Round} mph over the last 3 minutes on route: ${route}

You can refer to the input schema using EL to create customized alert messages

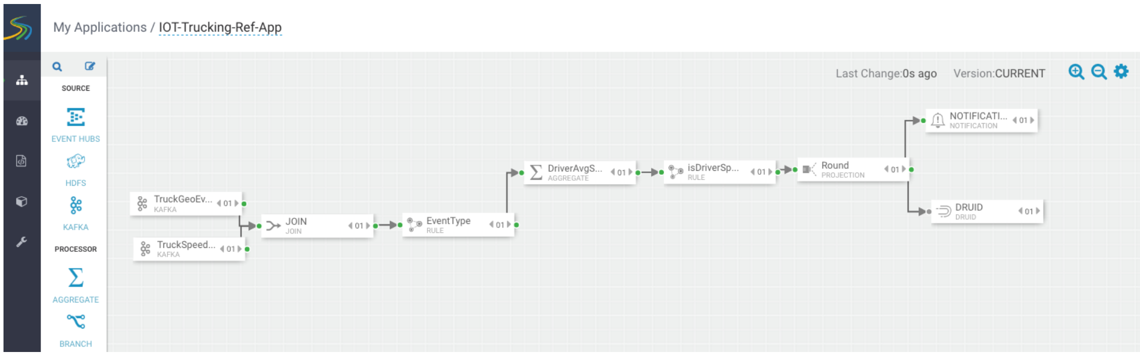
4.11. Streaming Alerts to an Analytics Engine for Dashboarding

About This Task

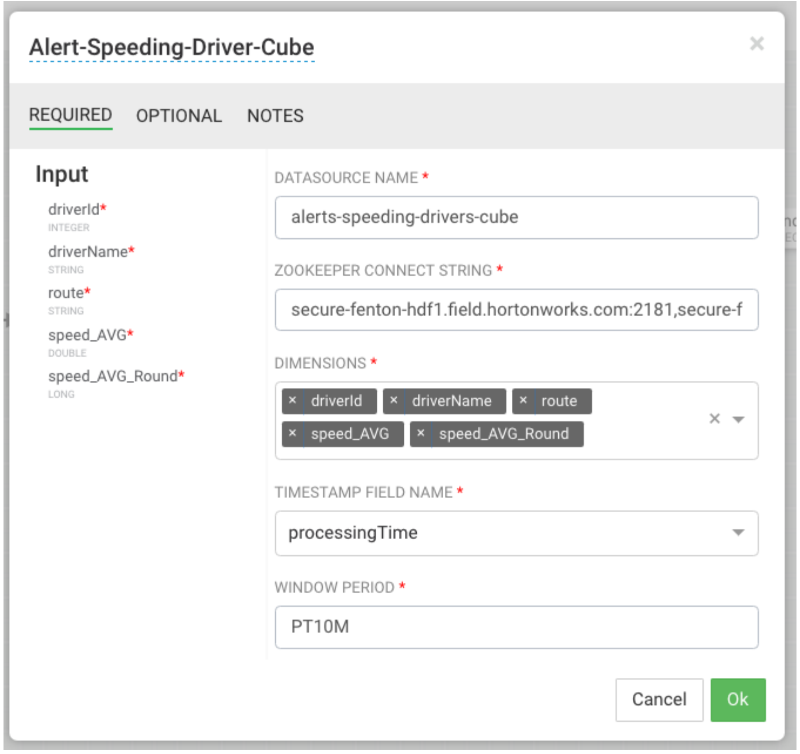
In addition creating notification alerts, a common use case requirement is to send these alerts to a dashboard so they can be displayed and visualized. SAM offers this capability by allowing you to stream data into DRUID and then using Superset to create dashboards and visualizations.

Steps

1. Drag the Druid sink to the canvas and connect to it to the Round Projection.



2. Stream these events into a Druid cube called **alerts-speeding-drivers-cube** by configuring the Druid processor like the following.



Alert-Speeding-Driver-Cube

Input

driverId* INTEGER

driverName* STRING

route* STRING

speed_AVG* DOUBLE

speed_AVG_Round* LONG

DATASOURCE NAME *: alerts-speeding-drivers-cube

ZOOKEEPER CONNECT STRING *: secure-fenton-hdf1.field.hortonworks.com:2181,secure-f

DIMENSIONS *: driverId, driverName, route, speed_AVG, speed_AVG_Round

TIMESTAMP FIELD NAME *: processingTime

WINDOW PERIOD *: PT10M

3. In the **Creating Visualization Section**, describe how to create dashboards for the **alerts-speeding-drivers-cube**.

4.12. Streaming Violation Events to an Analytics Engine for Descriptive Analytics

About This Task

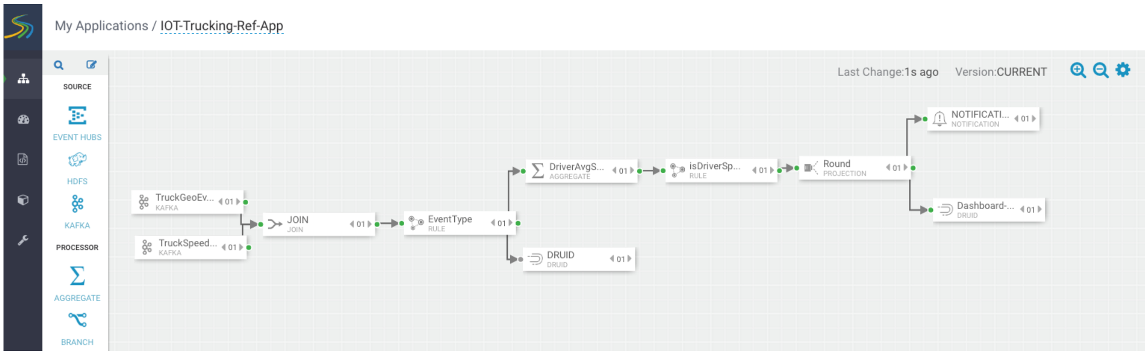
Now lets implement Requirement 7:

All infraction events need to be available for descriptive analytic (dash-boarding, visualizations, etc.) by a business analyst. The analyst needs the ability to do analysis on the streaming data.

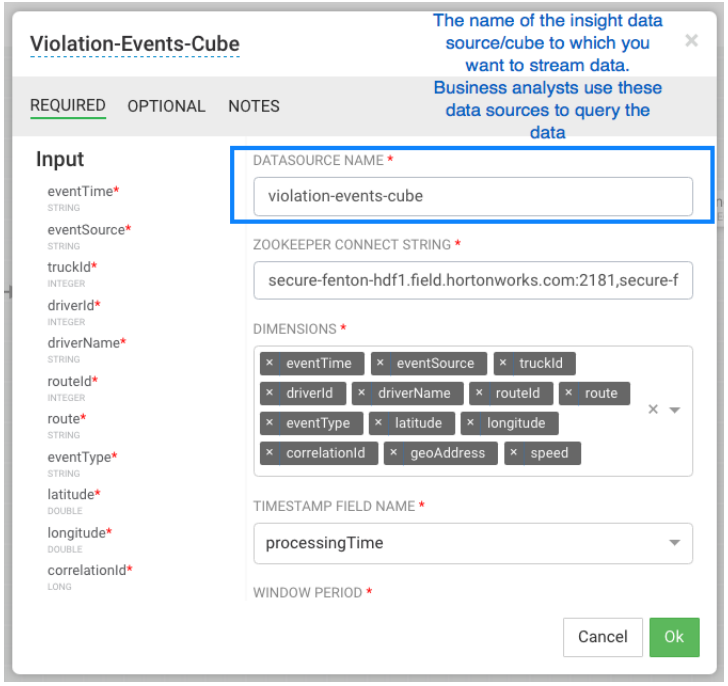
The analytics engine in SAM is powered by Druid. The following steps show how to stream data into Druid, so that a business analyst can use the Stream Insight Superset module to generate descriptive analytics.

Steps

1. Drag the Druid processor to the canvas and connect it to the ViolationEvents Rule processor.



2. Configure the Druid processor. You can edit the ZooKeeper connect string in the advanced section of the Druid Service in Ambari, under the property `druid.zk.service.host`.



The name of the insight data source/cube to which you want to stream data. Business analysts use these data sources to query the data

DATASOURCE NAME *
violation-events-cube

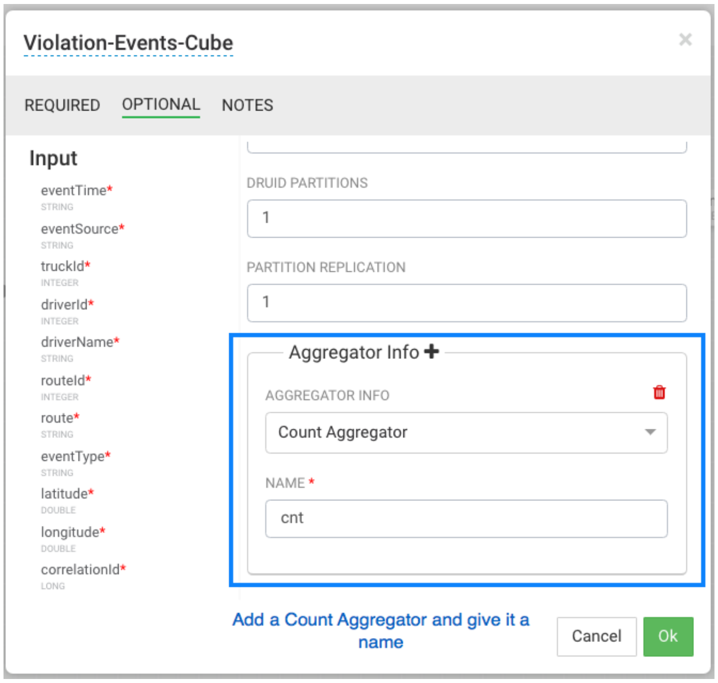
ZOOKEEPER CONNECT STRING *
secure-fenton-hdf1.field.hortonworks.com:2181,secure-f

DIMENSIONS *
eventTime, eventSource, truckId, driverId, driverName, routeId, route, eventType, latitude, longitude, correlationId, geoAddress, speed

TIMESTAMP FIELD NAME *
processingTime

WINDOW PERIOD *

3. Configure the **Aggregator Info** settings, under the **OPTIONAL** menu



Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

DRUID PARTITIONS: 1

PARTITION REPLICATION: 1

AGGREGATOR INFO: Count Aggregator

NAME *: cnt

Add a Count Aggregator and give it a name

4.13. Streaming Violation Events into a Data Lake and Operational Data Store

About This Task

Another common requirement is to stream data into an operational data store like HBase to power real-time web apps as well as a data lake powered by HDFS for long term storage and batch etl and analytic.

Steps

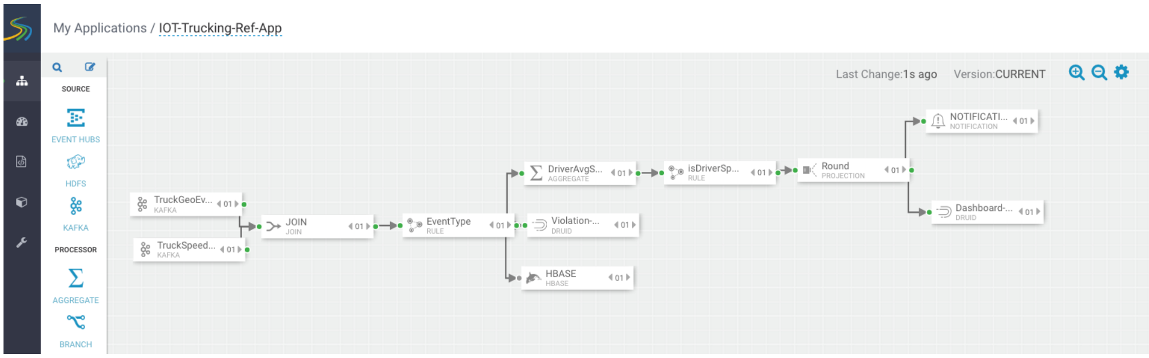
1. You will need ot have HBase service running. This can be easily done by adding the HDP HBase Service via Ambari. Create a new HBase table by logging into an node where Hbase client is installed then execute the below commands

   ```
   cd /usr/hdp/current/hbase-client/bin

   /hbase shell

   create 'violation_events', {NAME=> 'events', VERSIONS => 3} ;
   ```

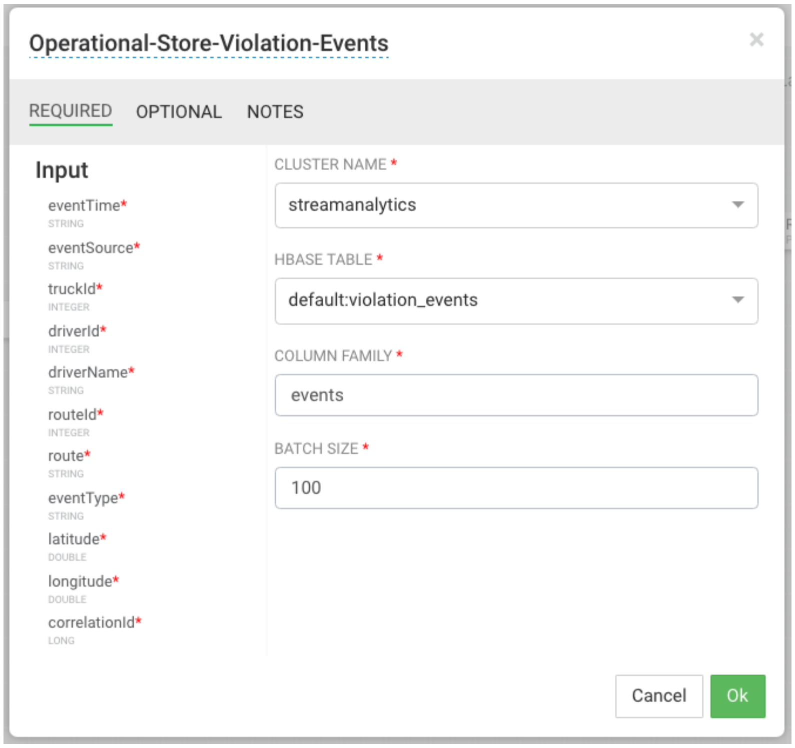
2. Create the following directory in HDFS and give it access to all users. Log into a node where HDFS client is installed and execute the below commands

   ```
   su hdfs

   hadoop fs -mkdir /apps/trucking-app

   hadoop fs -chmod 777 /apps/trucking-app
   ```

3. Drag the HBase sink to the canvas and connect it to the ViolationEvents Rule processor.



4. Configure the Hbase Sink as below.



Operational-Store-Violation-Events

REQUIRED OPTIONAL NOTES

Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

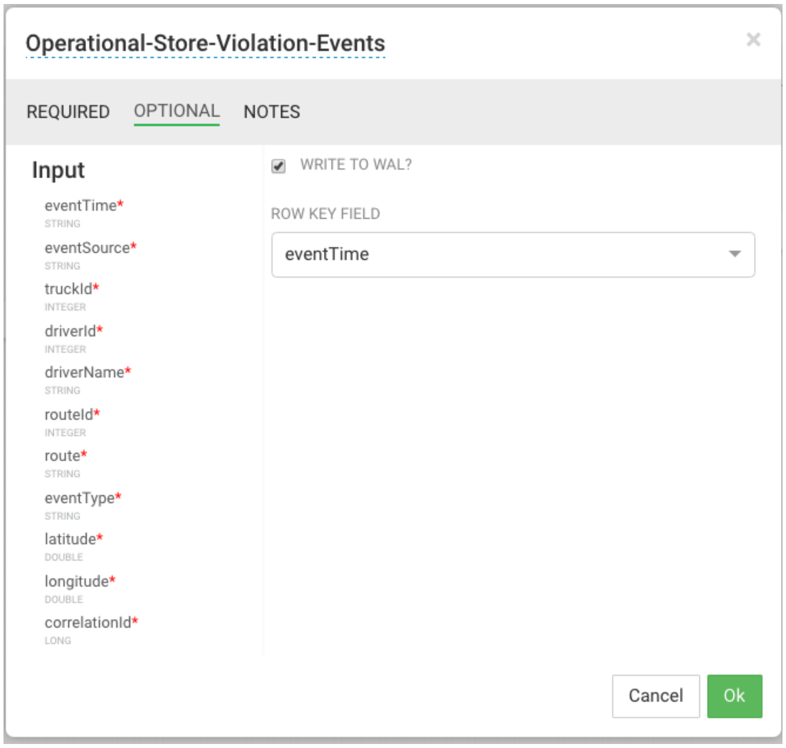
CLUSTER NAME *: streamanalytics

HBASE TABLE *: default:violation_events

COLUMN FAMILY *: events

BATCH SIZE *: 100

Cancel Ok



Operational-Store-Violation-Events

REQUIRED OPTIONAL NOTES

Input

eventTime*	STRING
eventSource*	STRING
truckId*	INTEGER
driverId*	INTEGER
driverName*	STRING
routeId*	INTEGER
route*	STRING
eventType*	STRING
latitude*	DOUBLE
longitude*	DOUBLE
correlationId*	LONG

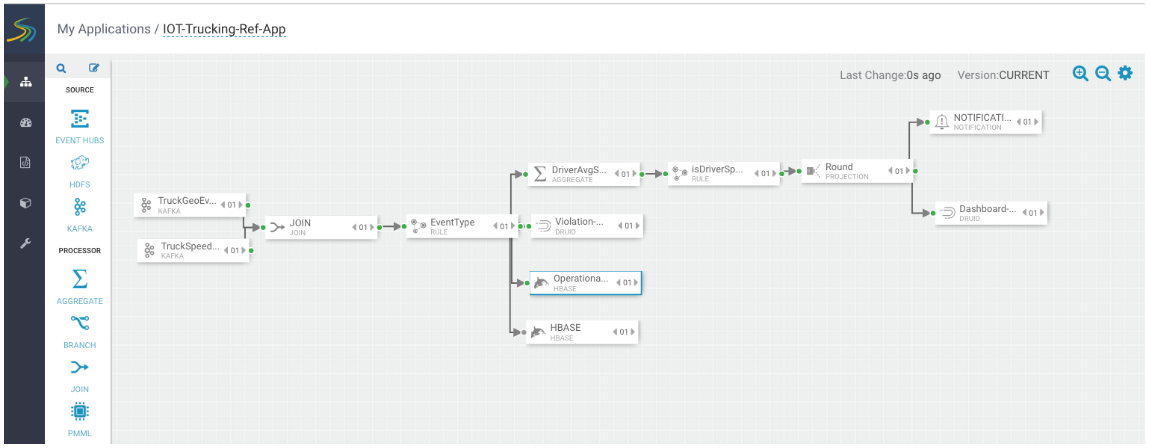
☑ WRITE TO WAL?

ROW KEY FIELD

eventTime

Cancel Ok

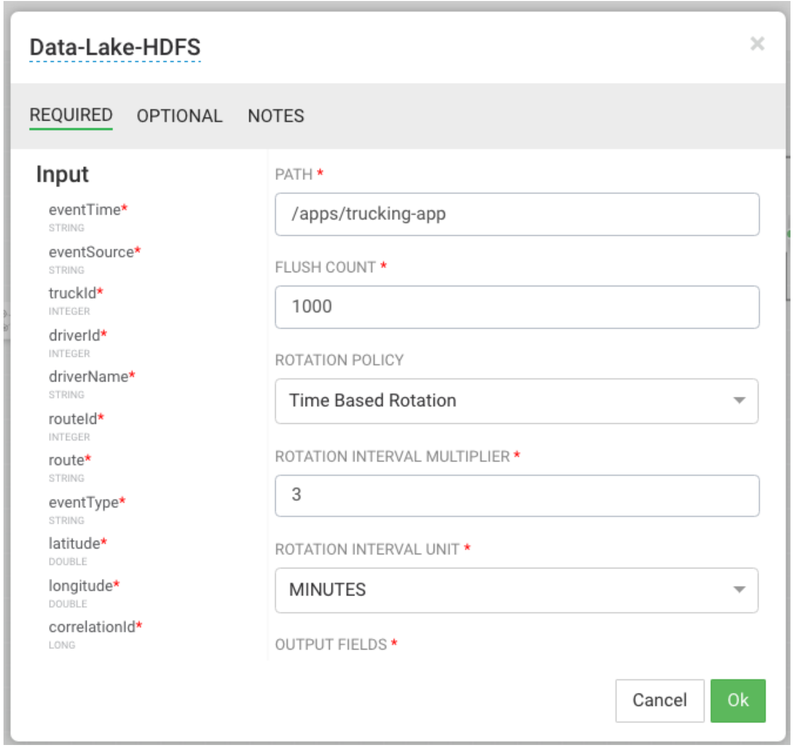
5. Drag the HDFS sink to the canvas and connect it to the ViolationEvents Rule processor.



My Applications / IOT-Trucking-Ref-App

Last Change:0s ago Version:CURRENT

6. Configure HDFS as below. Make sure you have permissiosn to write into the directory you have configired for HDFS path.



Data-Lake-HDFS

Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

PATH * /apps/trucking-app

FLUSH COUNT * 1000

ROTATION POLICY Time Based Rotation

ROTATION INTERVAL MULTIPLIER * 3

ROTATION INTERVAL UNIT * MINUTES

OUTPUT FIELDS *

5. Deploy an Application

5.1. Configure Deployment Settings

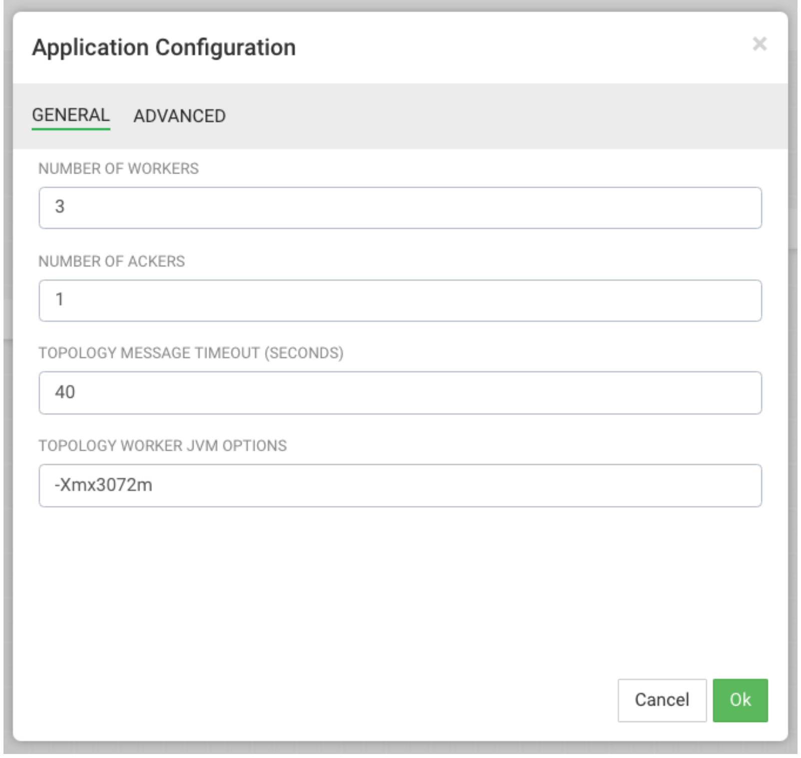
About This Task

Before deploying the application, you must configure deployment settings such as JVM size, number of ackers, and number of workers. Because this topology uses a number of joins and windows, you should increase the JVM heap size for the workers.

Steps

1. Click the gear icon on the top right corner of the canvas to display the **Application Configuration** dialog.
2. Increase **Number of Workers** to 5.
3. Set **Topology Worker JVM Options** to -Xmx3072m.

Example



Application Configuration

NUMBER OF WORKERS: 3

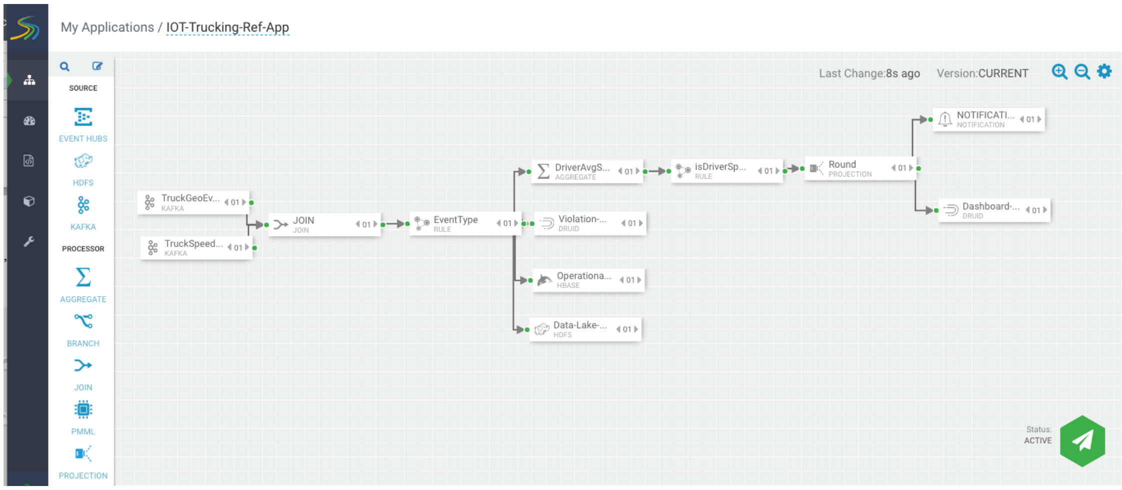
NUMBER OF ACKERS: 1

TOPOLOGY MESSAGE TIMEOUT (SECONDS): 40

TOPOLOGY WORKER JVM OPTIONS: -Xmx3072m

5.2. Deploy the App

After you have configure the application's deployment settings, click the **Deploy** button on the lower right of the canvas.



My Applications / IOT-Trucking-Ref-App

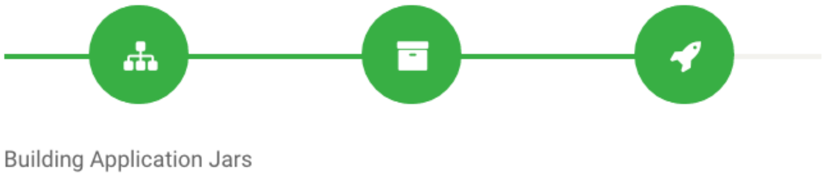
During the deployment process, Streaming Analytics Manager completes the following tasks:

1. Construct the configurations for the different big data services used in the stream app.

2. Create a deployable jar of the streaming app.

3. Upload and deploy the app jar to streaming engine server.

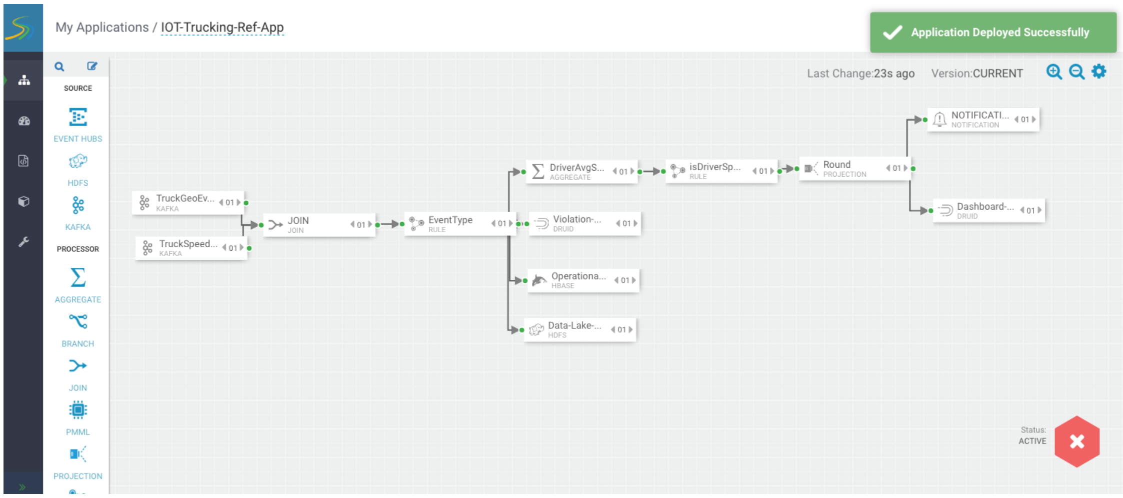
As SAM works through these tasks, it displays a progress bar.



Building Application Jars

The stream application is deployed to a Storm cluster based on the Storm Service defined in the Environment associated with the application.

After the application has been deployed successfully, SAM notifies you and updates the button to red to indicate it is deployed. Click the red button to kill/undeploy the app.



5.3. Running the Stream Simulator

Now that you have developed and deployed the NiFi Flow Application and the Stream Analytics Application, we are ready to run a data simulator that generates truck geo events and sensor events for the apps to process.

To generate the raw truck events serialized into Avro objects using the Schema registry and publish them into the raw Kafka topics, do the following:

1. Download the Data-Loader Unzip it and copy it to the node the cluster. Lets call the directory you unzip it to as: $DATA_LOADER_HOME. Then execute the following. Make sure to replace variables below with your environment specific values (you can find the REST URL to schema registry in Ambari under SAM service for config value registry.url) . Make sure java (jdk 1.8) is on your classpath.

```
tar -zxvf $DATA_LOADER_HOME/routes.tar.gz

nohup java -cp \
stream-simulator-jar-with-dependencies.jar \
hortonworks.hdp.refapp.trucking.simulator.
SimulationRegistrySerializerRunnerApp \
20000 \
hortonworks.hdp.refapp.trucking.simulator.impl.domain.transport.Truck \
hortonworks.hdp.refapp.trucking.simulator.impl.collectors.
KafkaEventSerializedWithRegistryCollector \
1 \
$DATA_LOADER_HOME/routes/midwest/ \
10000 \
$KAFKA_BROKER_HOST:$KAFKA_PORT \
$REST_URL_TO_SCHEMA_REGISTRY \
ALL_STREAMS \
NONSECURE &
```

2. You should see events being published into the Kafka topics called: raw-truck_events_avro and raw-truck_speed_events_avro, Nifi should be consuming those, enriching them and then pushing into the truck_events_avro and truck_speed_events_avro kafka topics and then SAM consumes from those topics.

6. Stream Operations

The Stream Operation view provides management of the stream applications, including the following:

- Application life cycle management: start, stop, edit, delete
- Application performance metrics
- Troubleshooting, debugging
- Exporting and importing applications

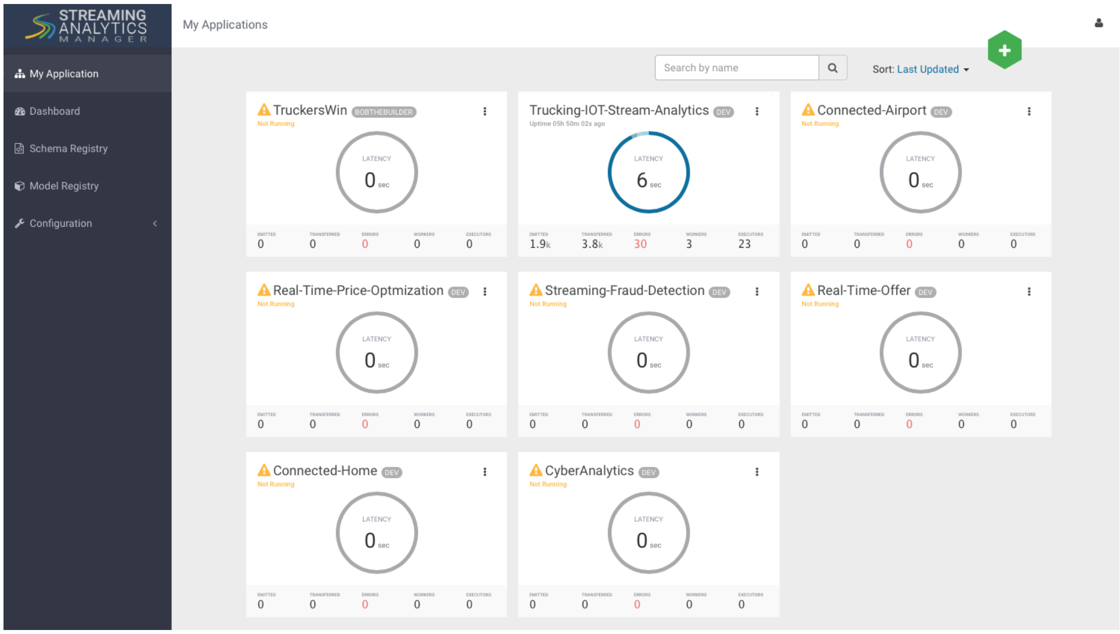
6.1. My Applications View

Once a stream application has been deployed, the Stream Operations displays operational views of the application.

One of these views is called **My Application** dashboard.

To access the application dashboard in SAM, click **My Application** tab (the hierarchy icon). The dashboard displays all applications built using Streaming Analytics Manager:

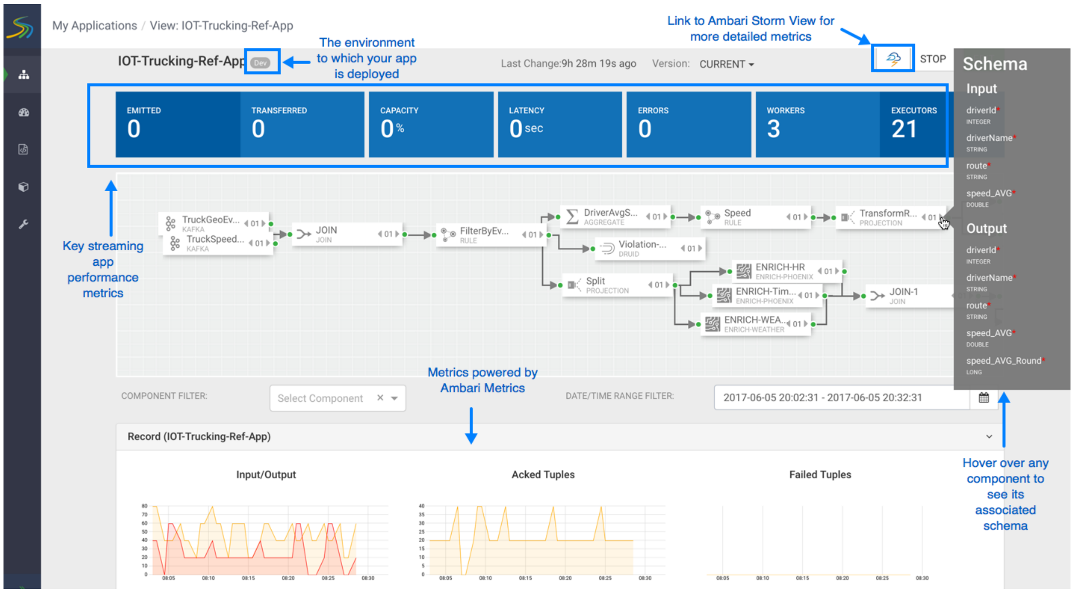
Each stream application is represented by an application tile. Hovering over the application tile provides status, metrics, and actions you can perform on the stream application.


STREAMING ANALYTICS MANAGER
My Application
Dashboard
Schema Registry
Model Registry
Configuration
My Applications
Search by name
Sort: Last Updated
TruckersWin BOBTHEBUILDER
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
Trucking-IOT-Stream-Analytics DEV
Uptime 05h 50m 02s ago
LATENCY 6 sec
EMITTED 1.9k TRANSFERRED 3.8k ERRORS 30 WORKERS 3 EXECUTORS 23
Connected-Airport DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
Real-Time-Price-Optmization DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
Streaming-Fraud-Detection DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
Real-Time-Offer DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
Connected-Home DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0
CyberAnalytics DEV
Not Running
LATENCY 0 sec
EMITTED 0 TRANSFERRED 0 ERRORS 0 WORKERS 0 EXECUTORS 0

6.2. Application Performance Monitoring

To view application performance metrics (APM) for the application, clicking on the application name on the application tile.

The following diagram describes elements of the APM view.



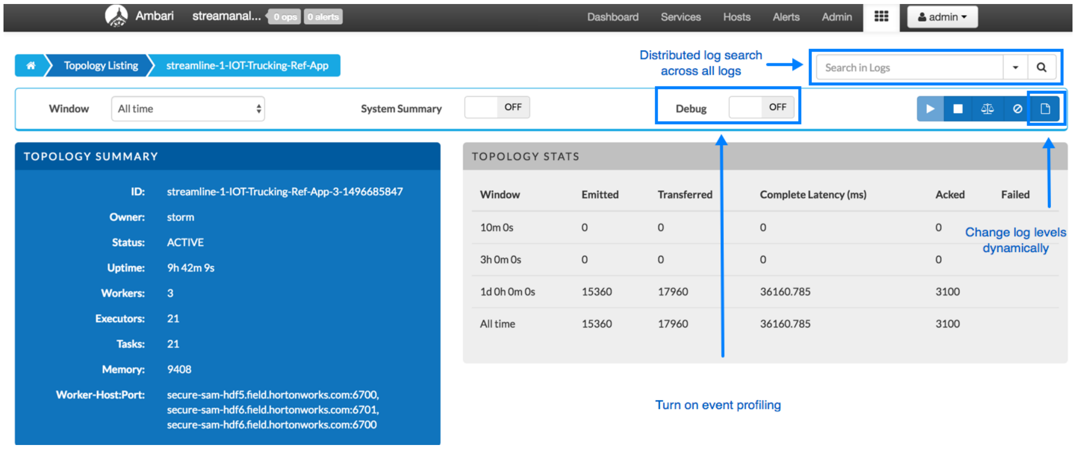
6.3. Troubleshooting and Debugging a Stream application

At the top right corner of the APM, there is a Storm icon that takes you to the Storm Ambari view.

The Storm Ambari View provides the following capabilities for deeper troubleshooting and debugging:

- Topology View and Metrics: shows a visual representation of the deployed topology and topology level Metrics.
- Distributed Log Search: allows users to search all logs across supervisor machines for a topology; results can include zipped logs.
- Dynamic Log Levels: allows Users and Administrators to dynamically change the log level settings for a running topology.
- Topology Event Inspector: allows viewing of tuples flowing through the topology along with the ability to turn on/off debug events without having to stop/restart the entire topology.
- Dynamic Worker Profiling: allows users to request worker profile data directly from the Storm UI (Heap Dumps, JStack Output, JProfile).

  Use the first portion of the Ambari Storm View to review the topology summary and statistics, set event profiling, search logs, and dynamically change them.



Distributed log search across all logs

Turn on event profiling

Change log levels dynamically

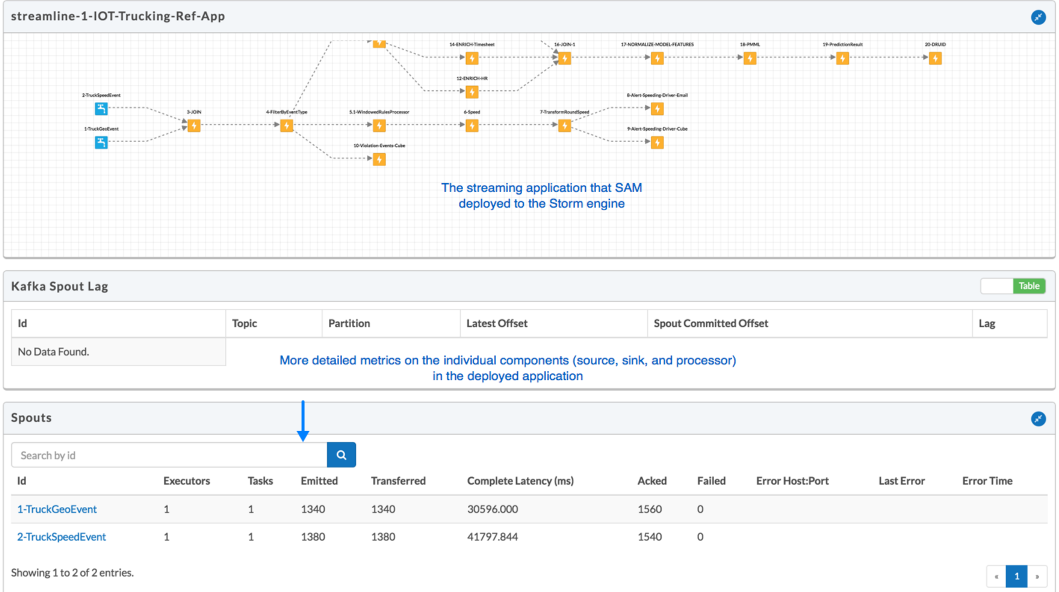
TOPOLOGY SUMMARY

ID:	streamline-1-IOT-Trucking-Ref-App-3-1496685847
Owner:	storm
Status:	ACTIVE
Uptime:	9h 42m 9s
Workers:	3
Executors:	21
Tasks:	21
Memory:	9408
Worker-Host:Port:	secure-sam-hdf5.field.hortonworks.com:6700, secure-sam-hdf6.field.hortonworks.com:6701, secure-sam-hdf6.field.hortonworks.com:6700

TOPOLOGY STATS

Window	Emitted	Transferred	Complete Latency (ms)	Acked	Failed
10m 0s	0	0	0	0	
3h 0m 0s	0	0	0	0	
1d 0h 0m 0s	15360	17960	36160.785	3100	
All time	15360	17960	36160.785	3100	

Scroll down to review the deployed topology and see metrics about its components.



The streaming application that SAM deployed to the Storm engine

Kafka Spout Lag

Id	Topic	Partition	Latest Offset	Spout Committed Offset	Lag
No Data Found.					

More detailed metrics on the individual components (source, sink, and processor) in the deployed application

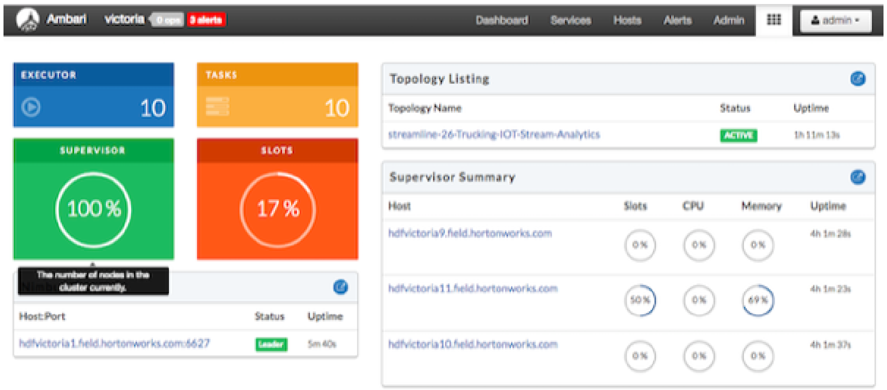
Spouts

Id	Executors	Tasks	Emitted	Transferred	Complete Latency (ms)	Acked	Failed	Error Host:Port	Last Error	Error Time
1-TruckGeoEvent	1	1	1340	1340	30596.000	1560	0			
2-TruckSpeedEvent	1	1	1380	1380	41797.844	1540	0			

Showing 1 to 2 of 2 entries.

6.3.1. Streaming Engine Infrastructure Metrics

The following dashboard shows infrastructure metrics for the streaming engine used; in this case, it shows details about the Storm cluster.



Host:Port	Status	Uptime
hdfvictoria1.field.hortonworks.com:6627	Leader	5m 40s

Topology Listing

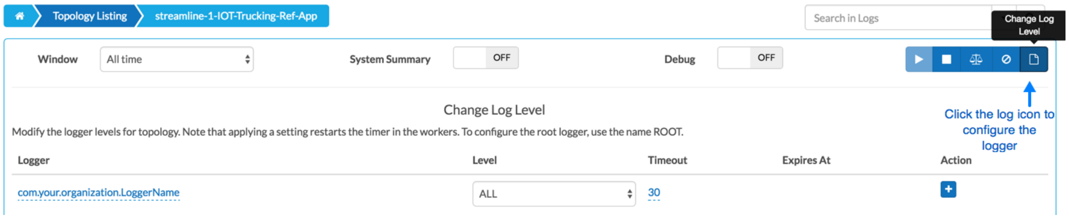
Topology Name	Status	Uptime
streamline-26-Trucking-IOT-Stream-Analytics	ACTIVE	1h 11m 13s

Supervisor Summary

Host	Slots	CPU	Memory	Uptime
hdfvictoria9.field.hortonworks.com	0 %	0 %	0 %	4h 1m 28s
hdfvictoria11.field.hortonworks.com	50 %	0 %	69 %	4h 1m 23s
hdfvictoria10.field.hortonworks.com	0 %	0 %	0 %	4h 1m 37s

6.3.2. Changing Log Levels Dynamically and with Expiration Policies

When debugging a stream application, the ability to change the log dynamically is a powerful troubleshooting feature. However, since typical stream applications handle millions of events per second, changes to log levels can impact performance unless safeguards such as expiration policies are defined. The following diagram shows how to change log levels with expiration policies.



Window: All time System Summary: OFF Debug: OFF

Click the log icon to configure the logger

Change Log Level

Modify the logger levels for topology. Note that applying a setting restarts the timer in the workers. To configure the root logger, use the name ROOT.

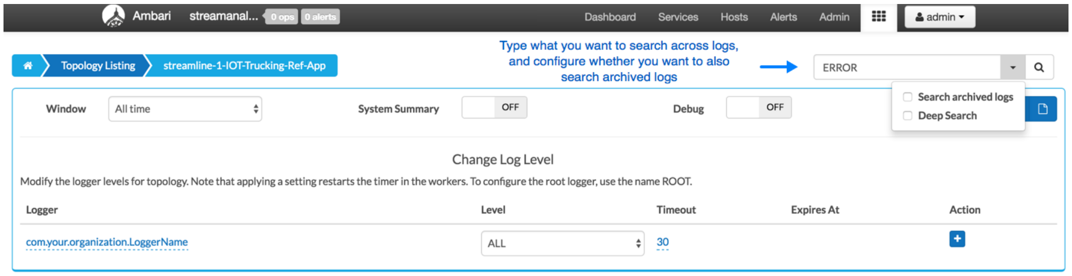
Logger	Level	Timeout	Expires At	Action
com.your.organization.LoggerName	ALL	30		+

6.3.3. Distributed Log Search

Storm is a distributed streaming engine, which means that many worker nodes can be used to power the streaming application. Because it has a distributed architecture, logs are distributed across the cluster on many worker nodes. Searching for log data across workers can be a painful process. With distributed log search, however, you can search across all logs located across all worker nodes.

The following steps describe how to use distributed log search.

1. Type your search string in the distributed log search text box:



Type what you want to search across logs, and configure whether you want to also search archived logs

ERROR

□ Search archived logs
□ Deep Search

Window: All time System Summary: OFF Debug: OFF

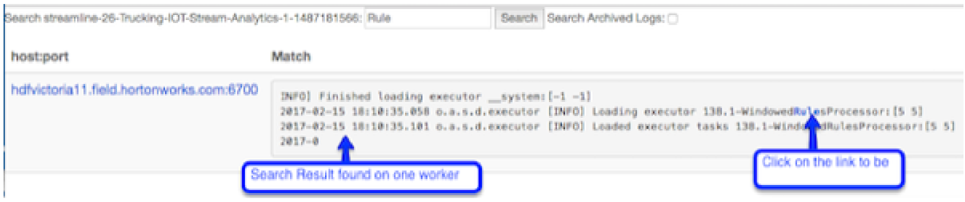
Change Log Level

Modify the logger levels for topology. Note that applying a setting restarts the timer in the workers. To configure the root logger, use the name ROOT.

Logger	Level	Timeout	Expires At	Action
com.your.organization.LoggerName	ALL	30		+

2. Review the results.

3. Click on the link to navigate to the exact location in the log file.



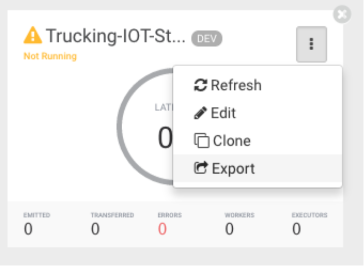
Search streamline-26-Trucking-IOT-Stream-Analytics-1-1487181566: Rule Search Search Archived Logs: □

host:port	Match
hdfvictoria11.field.hortonworks.com:6700	INFO] Finished loading executor __system:[-1 -1] 2017-02-15 18:10:35.058 o.a.s.d.executor [INFO] Loading executor 138.1-WindowedRulesProcessor:[5 5] 2017-02-15 18:10:35.101 o.a.s.d.executor [INFO] Loaded executor tasks 138.1-WindowedRulesProcessor:[5 5] 2017-0

6.4. Exporting and Importing Stream applications

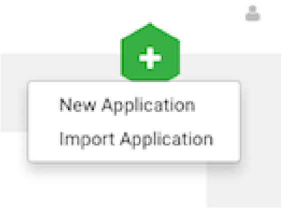
Service pool and environment abstractions combined with import and export capabilities allow you to move a stream application from one environment to another easily.

To export a stream application, click the Export icon on the **My Application** dashboard. This downloads a JSON file that represents your streaming application.

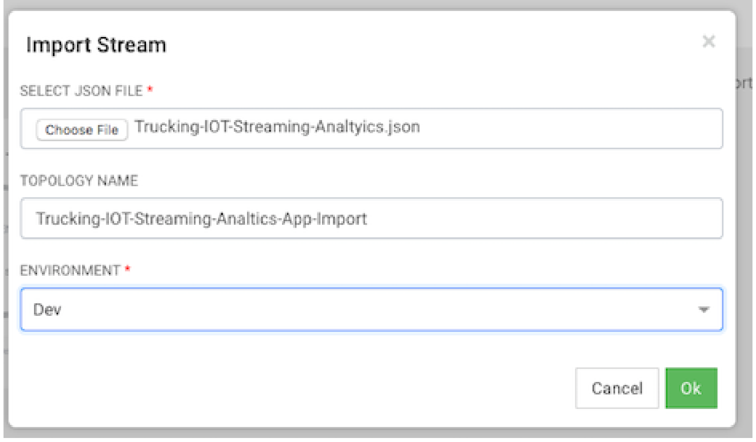


To import a stream application that was exported in JSON format:

1. Click on the + icon in **My Applications** View and select import application:



2. Select the JSON file that you want to import, provide a unique name for the application and specify which environment to use.



Import Stream

SELECT JSON FILE *

Choose File Trucking-IOT-Streaming-Analtyics.json

TOPOLOGY NAME

Trucking-IOT-Streaming-Analtics-App-Import

ENVIRONMENT *

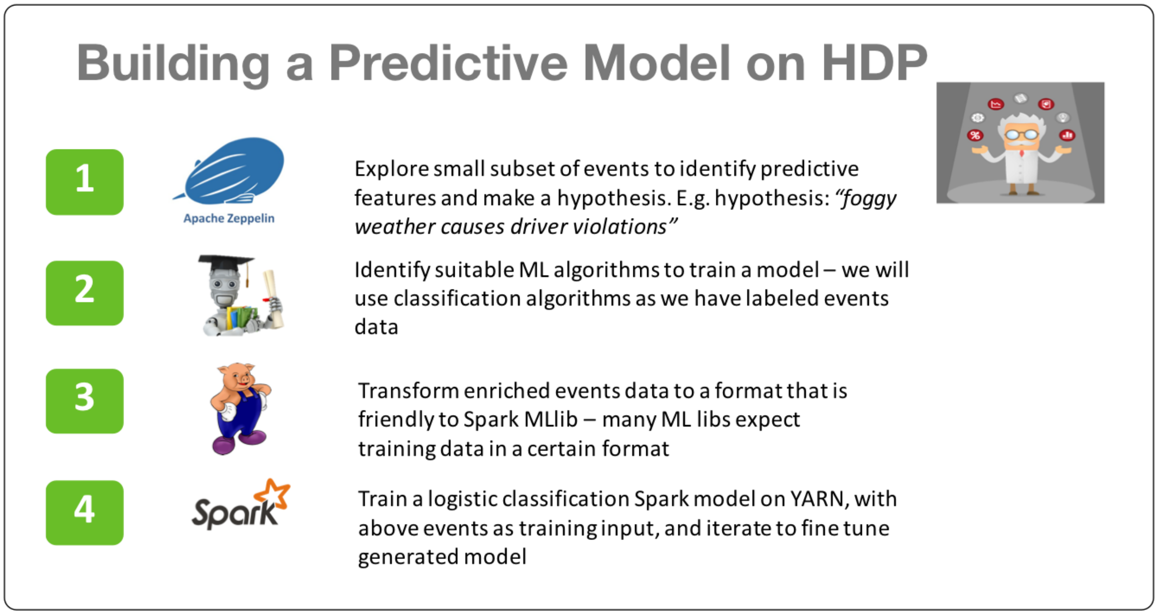
Dev

7. Advanced: Doing Predictive Analytics on the Stream

Requirement 10 of this use case states the following:

Execute a logistical regression Spark ML model on the events in the stream to predict if a driver is going to commit a violation. If violation is predicted, then alert on it.

HDP, the Hortonworks data at rest platform provides powerful set of tools for data engineers and scientists to build powerful analytics with data processing engines like Spark Streaming, Hive and Pig. The below diagram illustrates a typical analytics life cycle in HDP.



Apache Zeppelin
Spark

Building a Predictive Model on HDP

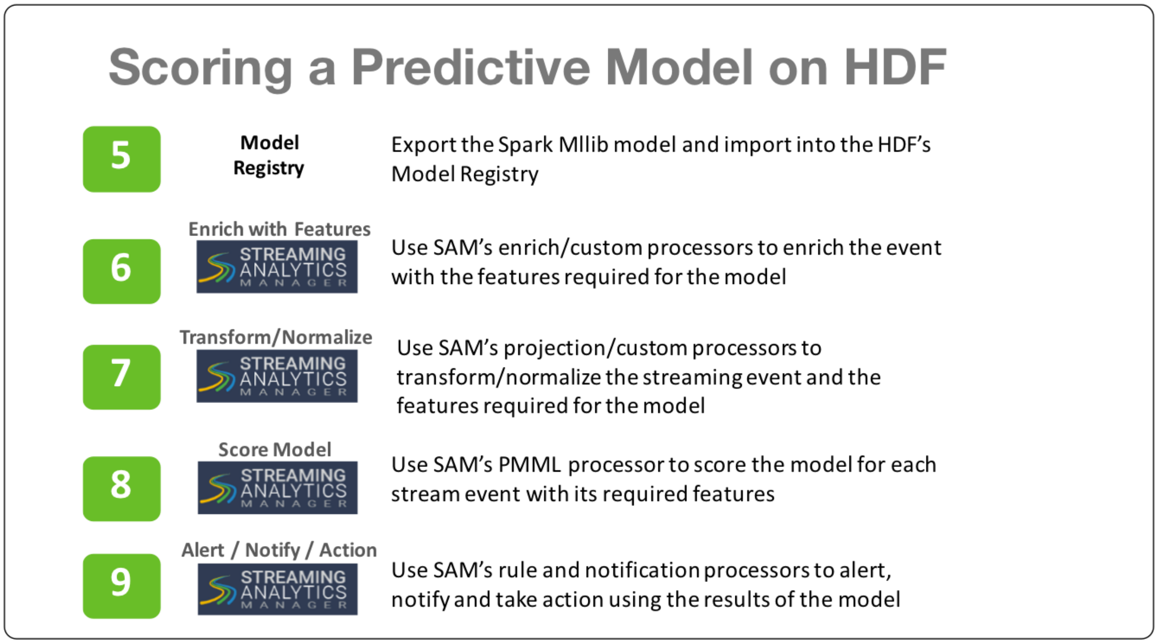
1 Explore small subset of events to identify predictive features and make a hypothesis. E.g. hypothesis: *"foggy weather causes driver violations"*

2 Identify suitable ML algorithms to train a model – we will use classification algorithms as we have labeled events data

3 Transform enriched events data to a format that is friendly to Spark MLlib – many ML libs expect training data in a certain format

4 Train a logistic classification Spark model on YARN, with above events as training input, and iterate to fine tune generated model

Once the model has been trained and optimized, insights can be created by scoring the model in real-time as events are coming in. The next set of steps in the life cycle has to do with scoring the model in real-time using HDF components.



Scoring a Predictive Model on HDF

5 **Model Registry** — Export the Spark Mllib model and import into the HDF's Model Registry

6 **Enrich with Features** — Use SAM's enrich/custom processors to enrich the event with the features required for the model

7 **Transform/Normalize** — Use SAM's projection/custom processors to transform/normalize the streaming event and the features required for the model

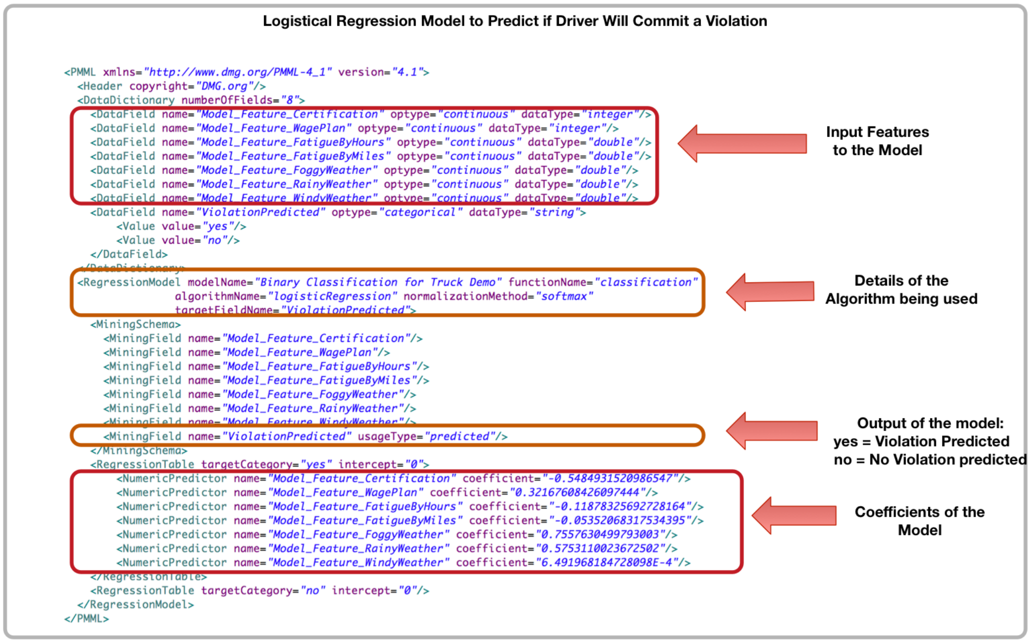
8 **Score Model** — Use SAM's PMML processor to score the model for each stream event with its required features

9 **Alert / Notify / Action** — Use SAM's rule and notification processors to alert, notify and take action using the results of the model

In the next few sections we will walk through how to do steps 5 through 9 in SAM.

7.1. Logistical Regression Model

With steps 1-4 with HDP, we were able to build a logistical regression model. The model was then exported into PMML. The below diagram illustrates the features, coefficients and output of the model.



Logistical Regression Model to Predict if Driver Will Commit a Violation

```
<PMML xmlns="http://www.dmg.org/PMML-4_1" version="4.1">
  <Header copyright="DMG.org"/>
  <DataDictionary numberOfFields="8">
    <DataField name="Model_Feature_Certification" optype="continuous" dataType="integer"/>
    <DataField name="Model_Feature_WagePlan" optype="continuous" dataType="integer"/>
    <DataField name="Model_Feature_FatigueByHours" optype="continuous" dataType="double"/>
    <DataField name="Model_Feature_FatigueByMiles" optype="continuous" dataType="double"/>
    <DataField name="Model_Feature_FoggyWeather" optype="continuous" dataType="double"/>
    <DataField name="Model_Feature_RainyWeather" optype="continuous" dataType="double"/>
    <DataField name="Model_Feature_WindyWeather" optype="continuous" dataType="double"/>
    <DataField name="ViolationPredicted" optype="categorical" dataType="string">
        <Value value="yes"/>
        <Value value="no"/>
    </DataField>
  </DataDictionary>
  <RegressionModel modelName="Binary Classification for Truck Demo" functionName="classification"
                algorithmName="logisticRegression" normalizationMethod="softmax"
                targetFieldName="ViolationPredicted">
    <MiningSchema>
      <MiningField name="Model_Feature_Certification"/>
      <MiningField name="Model_Feature_WagePlan"/>
      <MiningField name="Model_Feature_FatigueByHours"/>
      <MiningField name="Model_Feature_FatigueByMiles"/>
      <MiningField name="Model_Feature_FoggyWeather"/>
      <MiningField name="Model_Feature_RainyWeather"/>
      <MiningField name="Model_Feature_WindyWeather"/>
      <MiningField name="ViolationPredicted" usageType="predicted"/>
    </MiningSchema>
    <RegressionTable targetCategory="yes" intercept="0">
        <NumericPredictor name="Model_Feature_Certification" coefficient="-0.5484931520986547"/>
        <NumericPredictor name="Model_Feature_WagePlan" coefficient="0.32167608426097444"/>
        <NumericPredictor name="Model_Feature_FatigueByHours" coefficient="-0.11878325692728164"/>
        <NumericPredictor name="Model_Feature_FatigueByMiles" coefficient="-0.05352068317534395"/>
        <NumericPredictor name="Model_Feature_FoggyWeather" coefficient="0.7557630499793003"/>
        <NumericPredictor name="Model_Feature_RainyWeather" coefficient="0.5753110023672502"/>
        <NumericPredictor name="Model_Feature_WindyWeather" coefficient="6.491968184728098E-4"/>
    </RegressionTable>
    <RegressionTable targetCategory="no" intercept="0"/>
  </RegressionModel>
</PMML>
```

Input Features to the Model

Details of the Algorithm being used

Output of the model:
yes = Violation Predicted
no = No Violation predicted

Coefficients of the Model

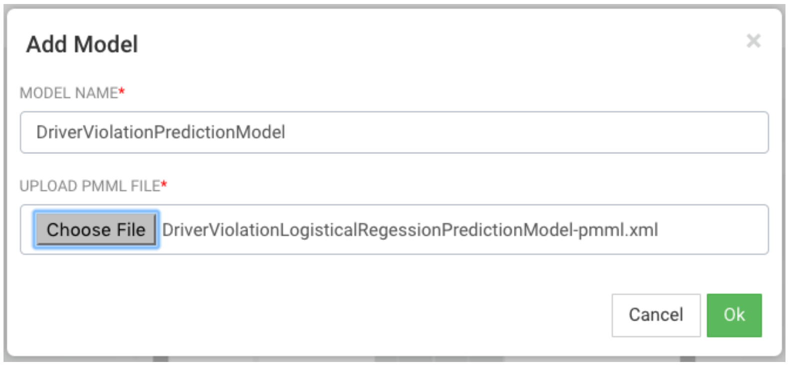
7.2. Export the Model into SAM's Model Registry

About This Task

SAM provides a registry where you can store PMML models. To get started with predictive analytics, upload this logistical regression model.

Steps

1. Download this PMML model and save it locally with an `.xml` extension.
2. Select the **Model Registry** menu item.
3. Click the + icon.
4. Give your PMML model a name.
5. From **Upload PMML File**, select the PMML file you just downloaded.



Add Model

MODEL NAME*

DriverViolationPredictionModel

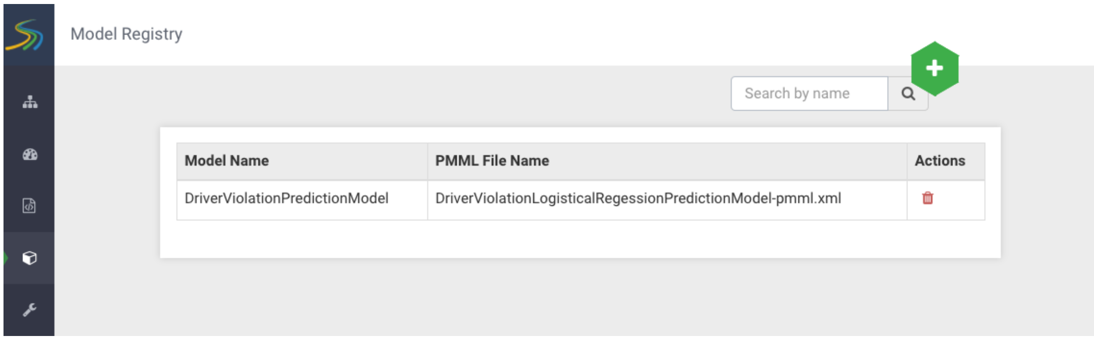
UPLOAD PMML FILE*

Choose File DriverViolationLogisticalRegessionPredictionModel-pmml.xml

6. Click **Ok**.

Result

The model is saved in the **Model Registry**.



Model Registry

Model Name	PMML File Name	Actions
DriverViolationPredictionModel	DriverViolationLogisticalRegessionPredictionModel-pmml.xml	🗑

7.3. Enrichment and Normalization of Model Features

Now that the model has been added to the model registry, you can use it in the streaming application by the PMML processor. Before the model can be executed, you must enrich and normalize the streaming events with the features required by the model. As the above diagram illustrates, there are 7 features in the model. None of these features come as part of the stream from the two sensors. So, based on the driverId and the latitude and longitude location, enrich the streaming event with these features and then normalize it required by the model. The table below describe each feature, enrichment store, and the normalization required.

Feature	Description	Enrichment Store	Normalization
Model_Feature_Certification	Identifies if the driver is certified or not	HBase/Phoenix table called drivers	"yes" # normalize to 1 "no" # normalize to 0
Model_Feature_WagePlan	Identifies if the driver is on an hourly or by miles wage plan	HBase/Phoenix table called drivers	"Hourly" # normalize to 1 "Miles" # normalize to 0
Model_Feature_FatigueByHours	The total number of hours driven by the driver in the last week	HBase/Phoenix table called timesheet	Scale by 100 to improve algorithm performance (e.g: hours/100)
Model_Feature_FatigueByMiles	The total number of miles driven by the driver in the last week	HBase/Pheonix table called timesheet	Scale by 1000 to improve algorithm performance (e.g: miles/1000)
Model_Feature_FoggyWeather	Determines if for the given time and location, if the conditions are foggy	API to WeatherService	if (foggy) # normalize to 1 else 0
Model_Feature_RainyWeather	Determines if for the given time and location, if the conditions are rainy	API to WeatherService	if (raining) –> normalize to 1 else 0
Model_Feature_WindyWeather	Determines if for the given time and location, if the conditions are windy	API to WeatherService	if (windy) # normalize to 1 else 0

7.4. Setting up your Enrichment Store and Building Custom UDFs and Processors

About This Task

As the table above indicate four of the seven features comes from HBase/Phoenix tables. This section gives you instructions on setting up the HBase/Phonenix tables timesheet and drivers, loading them with reference data, and downloading the custom UDFs and processors to do the enrichment and normalization.

Install HBase/Phoenix and download the sam-extensions

1. If HBase is not installed, install/add an HBase service.

2. Ensure that Phoenix is enabled on the HBase Cluster.

3. Download the Sam-Custom-Extensions.zip and save it to your local machine.

4. Unzip the contents. We will call the unziped folder $SAM_EXTENSIONS

Steps for Creating Phoenix Tables and Loading Reference Data

1. Copy the $SAM_EXTENSIONS/scripts.tar.gz to a node where where HBase/PHoenix client is installed.

2. On that node, untar the scripts.tar.gz. We will call this directory $SCRIPTS

   ```
   tar -zxvf scripts.tar.gz
   ```

3. Go to the directory where the phoenix script is located which will create the phoeneix tables for enrichment and load it with reference data.

   ```
   cd $SCRIPTS/phoenix
   ```

4. Open the file phoenix_create.sh and replace <ZK_HOST> with the FQDN of your ZooKeeper host.

5. Make the phoenix_create.sh script executable and execute it. Make sure you to JAVA_HOME

   ```
   ./phoenix_create.sh
   ```

Steps for Verifying Data has Populated Phoenix Tables

1. Start up sqline Phoenix client.

   ```
   cd /usr/hdp/current/phoenix-client/bin

   ./sqlline.py $ZK_HOST:2181:/hbase-unsecure
   ```

2. List all the tables in Pheonix.

   ```
   !tables
   ```

3. Query the drivers and timesheet tables.

   ```
   select * from drivers;
   select * from timesheet;
   ```

7.5. Upload Custom Processors and UDFs for Enrichment and Normalization

To perform the above enrichment and normalization, upload the custom UDFs and processors you downloaded in the previous section.

7.5.1. Upload Custom UDFs

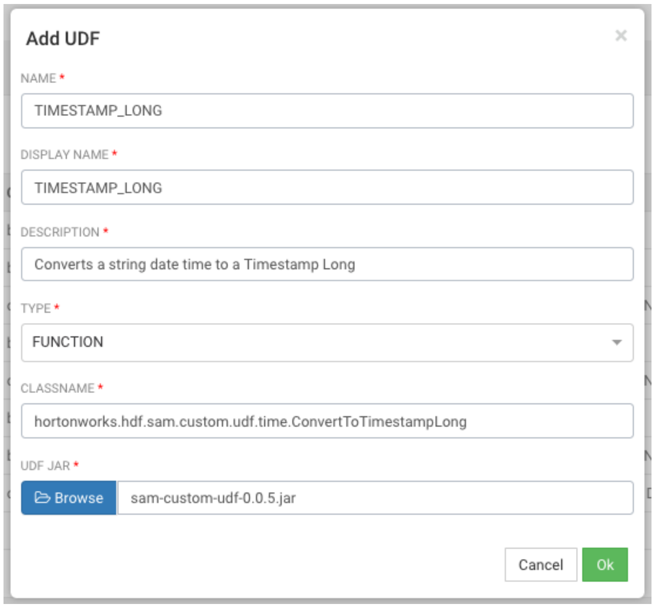
Steps for Uploading the Timestamp_Long UDF

1. From the left-hand menu, click **Configuration**, then **Application Resources**.

2. Select the **UDF** Tab and click the + sign to create the TIMESTAMP_LONG UDF. This udf will convert a string date time to a Timestamp long. The simple class for this UDF using the SAM SDK can be found here.

3. The jar for this UDF is located in `SAM_EXTENSIONS/sam-custom-udf-0.0.5.jar`.

4. Configure the UDF with the following values:



Add UDF

NAME *: TIMESTAMP_LONG

DISPLAY NAME *: TIMESTAMP_LONG

DESCRIPTION *: Converts a string date time to a Timestamp Long

TYPE *: FUNCTION

CLASSNAME *: hortonworks.hdf.sam.custom.udf.time.ConvertToTimestampLong

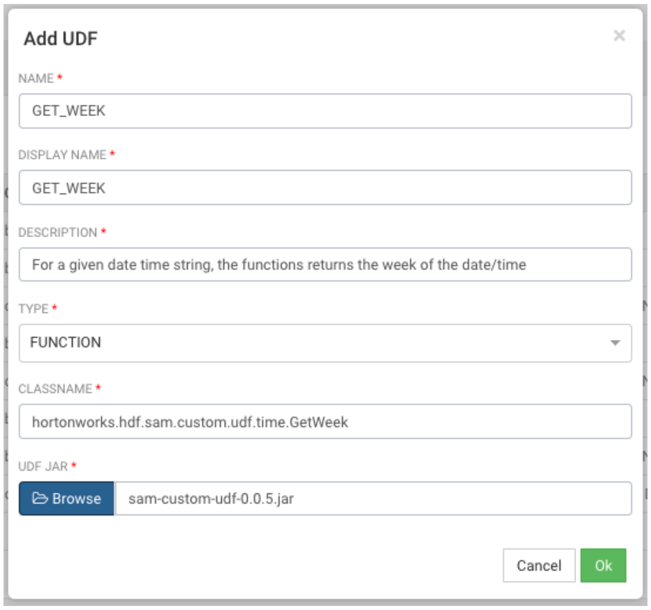
UDF JAR *: sam-custom-udf-0.0.5.jar

Steps for Configuring the Get_Week UDF

1. Select the **UDF** tab and click the + sign to create the GET_WEEK UDF.

2. The `jar` for this UDF is located in `SAM_EXTENSIONS/sam-custom-udf-0.0.5.jar`. This udf will convert a timestamp string into the week of the year which is required for an enrichment query. The simple class for this UDF using the SAM SDK can be found here.

3. Configure the UDF with the following values:



Add UDF

NAME *: GET_WEEK

DISPLAY NAME *: GET_WEEK

DESCRIPTION *: For a given date time string, the functions returns the week of the date/time

TYPE *: FUNCTION

CLASSNAME *: hortonworks.hdf.sam.custom.udf.time.GetWeek

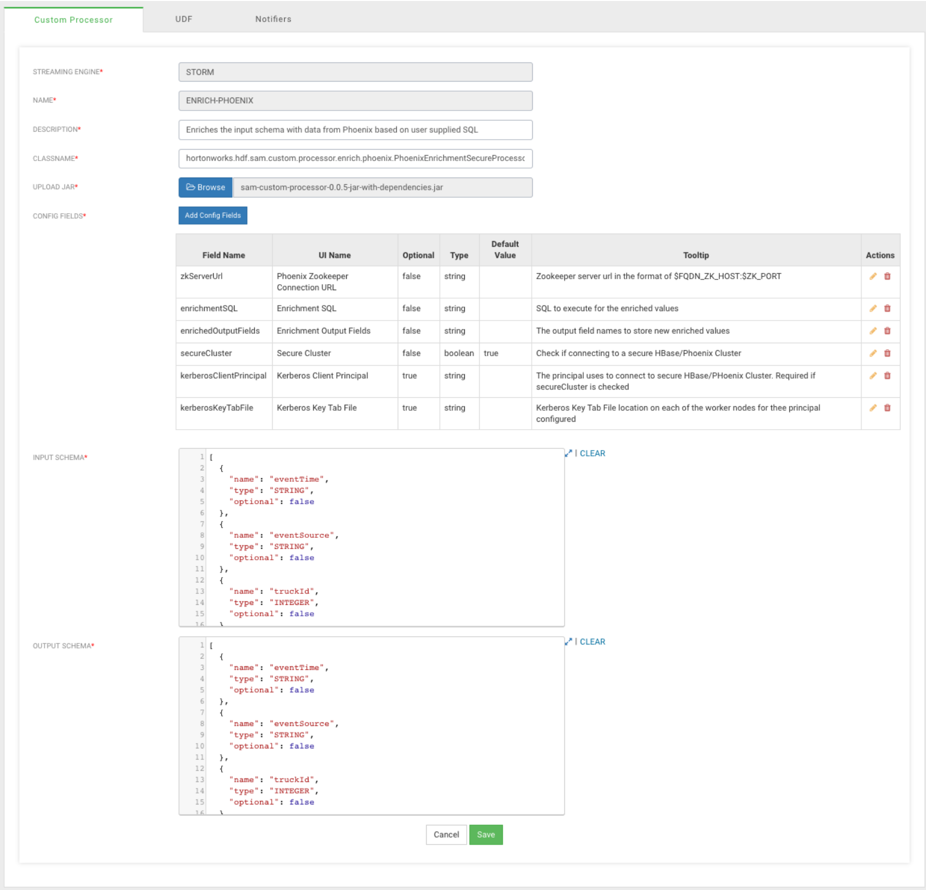
UDF JAR *: sam-custom-udf-0.0.5.jar

7.5.2. Upload Custom Processors

Steps for Uploading the ENRICH-PHOENIX Custom Processor

1. From the left-hand menu, click **Configuration**, then **Application Resources**.

2. Select **Custom Processor** and click the + sign to create the ENRICH-PHOENIX processor. Configure the processor with the following values. This processor can be used to enriched streams with data from Phoenix based on a user supplied sql statement. The java class for this processor using the SAM SDK can be found here.



STREAMING ENGINE*	STORM
NAME*	ENRICH-PHOENIX
DESCRIPTION*	Enriches the input schema with data from Phoenix based on user supplied SQL
CLASSNAME*	hortonworks.hdf.sam.custom.processor.enrich.phoenix.PhoenixEnrichmentSecureProcess
UPLOAD JAR*	Browse sam-custom-processor-0.0.5-jar-with-dependencies.jar
CONFIG FIELDS*	Add Config Fields

Field Name	UI Name	Optional	Type	Default Value	Tooltip	Actions
zkServerUrl	Phoenix Zookeeper Connection URL	false	string		Zookeeper server url in the format of $FQDN_ZK_HOST:$ZK_PORT	✎ 🗑
enrichmentSQL	Enrichment SQL	false	string		SQL to execute for the enriched values	✎ 🗑
enrichedOutputFields	Enrichment Output Fields	false	string		The output field names to store new enriched values	✎ 🗑
secureCluster	Secure Cluster	false	boolean	true	Check if connecting to a secure HBase/Phoenix Cluster	✎ 🗑
kerberosClientPrincipal	Kerberos Client Principal	true	string		The principal uses to connect to secure HBase/PHoenix Cluster. Required if secureCluster is checked	✎ 🗑
kerberosKeyTabFile	Kerberos Key Tab File	true	string		Kerberos Key Tab File location on each of the worker nodes for thee principal configured	✎ 🗑

INPUT SCHEMA*

```
[
  {
    "name": "eventTime",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "eventSource",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "truckId",
    "type": "INTEGER",
    "optional": false
```

OUTPUT SCHEMA*

```
[
  {
    "name": "eventTime",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "eventSource",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "truckId",
    "type": "INTEGER",
    "optional": false
```

ENRICH-PHOENIX Configuration Values

- **Streaming Engine** – Storm
- **Name** – ENRICH-PHOENIX
- **Description** – Enriches the input schema with data from Phoenix based on user supplied SQL
- **ClassName** – `hortonworks.hdf.sam.custom.processor.enrich.phoenix.PhoenixEnrichmentSecure`

- **Upload Jar** – The jar for this custom processor can be found under `SAM_EXTENSIONS/sam-custom-processor-0.0.5-jar-with-dependencies.jar`

Click the **Add Config Fields** button and and the following 3 configuration fields

- Add a config field called **zkServerUrl** with the following values:

  a. **Field Name** – zkServerUrl

  b. **UI Name** – Phoenix ZooKeeper Connection URL

  c. **Optional** – false

  d. **Type** – string

  e. **ToolTip** – ZooKeeper server url in the format of $FQDN_ZK_HOST:$ZK_PORT

- Add a config field called **enrichmentSQL** with the following values

  a. **Field Name** – enrichmentSQL

  b. **UI Name** – Enrichment SQL

  c. **Optional** – false

  d. **Type** – string

  e. **ToolTip** – SQL to execute for the enriched values

- Add a config field called **enrichedOutputFields** with the following values:

  a. **Field Name** – enrichedOutputFields

  b. **UI Name** – Enrichment Output Fields

  c. **Optional** – false

  d. **Type** – string

  e. **ToolTip** – The output field names to store new enriched values

- Add a config field called **secureCluster** with the following values:

  a. **Field Name** – secureCluster

  b. **UI Name** – Secure Cluster

  c. **Optional** – false

  d. **Type** – boolean

  e. **ToolTip** – Check if connecting to a secure HBase/Phoenix Cluster

- Add a config field called **kerberosClientPrincipal** with the following values:

  a. **Field Name** – kerberosClientPrincipal

  b. **UI Name** – Kerberos Client Principal

  c. **Optional** – true

  d. **Type** – string

  e. **ToolTip** – The principal uses to connect to secure HBase/PHoenix Cluster. Required if secureCluster is checked

- Add a config field called **kerberosKeyTabFile** with the following values:

  a. **Field Name** – kerberosKeyTabFile

  b. **UI Name** – Kerberos Key Tab File

  c. **Optional** – true

  d. **Type** – string

  e. **ToolTip** – Kerberos Key Tab File location on each of the worker nodes for thee principal configured

Input and Output Schema for ENRICH-PHOENIX

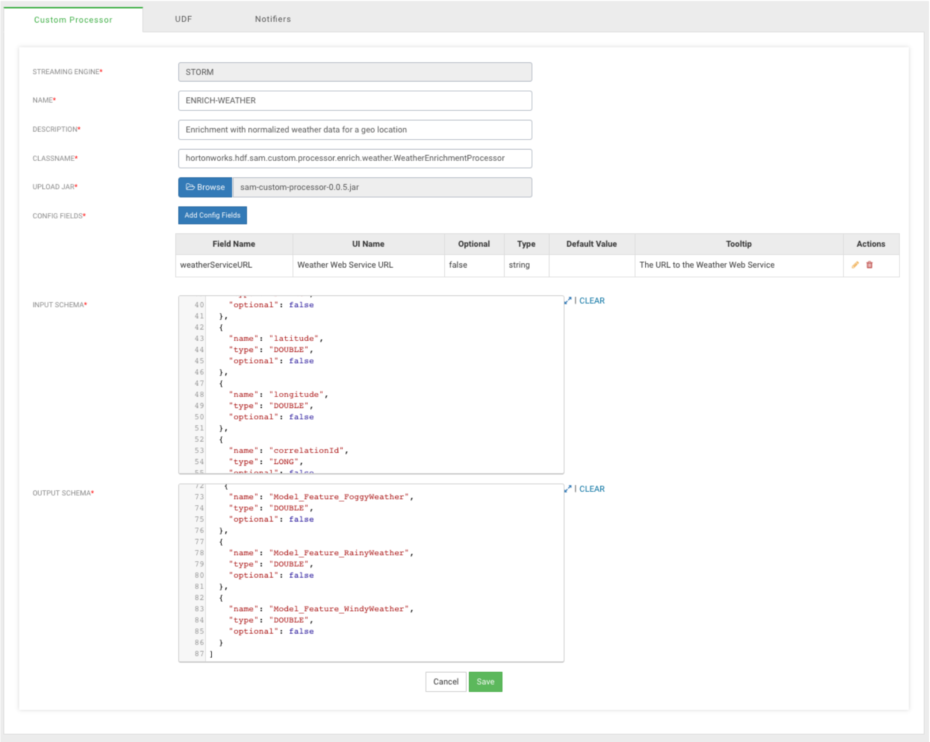
- Copy this input schema and paste into the INPUT SCHEMA text area box

- Copy this output schema and paste into the OUTPUT SCHEMA text area box

Steps for Uploading the ENRICH-WEATHER Custom Processor

1. Select **Custom Processor** and click the + sign to create the ENRICH-WEATHER processor. This processor can be used to enrich streams with weather data based on time and lat/long location. The java class for this processor using the SAM SDK can be found here.

2. Configure the processor with the following values.



STREAMING ENGINE* STORM

NAME* ENRICH-WEATHER

DESCRIPTION* Enrichment with normalized weather data for a geo location

CLASSNAME* hortonworks.hdf.sam.custom.processor.enrich.weather.WeatherEnrichmentProcessor

UPLOAD JAR* sam-custom-processor-0.0.5.jar

CONFIG FIELDS*

Field Name	UI Name	Optional	Type	Default Value	Tooltip	Actions
weatherServiceURL	Weather Web Service URL	false	string		The URL to the Weather Web Service	

INPUT SCHEMA*

```
    "optional": false
  },
  {
    "name": "latitude",
    "type": "DOUBLE",
    "optional": false
  },
  {
    "name": "longitude",
    "type": "DOUBLE",
    "optional": false
  },
  {
    "name": "correlationId",
    "type": "LONG",
```

OUTPUT SCHEMA*

```
    "name": "Model_Feature_FoggyWeather",
    "type": "DOUBLE",
    "optional": false
  },
  {
    "name": "Model_Feature_RainyWeather",
    "type": "DOUBLE",
    "optional": false
  },
  {
    "name": "Model_Feature_WindyWeather",
    "type": "DOUBLE",
    "optional": false
  }
]
```

ENRICH-WEATHER Configuration Values

- **Streaming Engine** – Storm
- **Name** – ENRICH-WEATHER
- **Description** – Enrichment with normalized weather data for a geo location
- **ClassName** – hortonworks.hdf.sam.custom.processor.enrich.weather.WeatherEnrichmentProcessor
- **Upload Jar** – The jar for this custom processor can be found under `SAM_EXTENSIONS/sam-custom-processor-0.0.5.jar`

Click the **Add Config Fields** button and and a configuration field with the following values:

- **Field Name** – weatherServiceURL
- **UI Name** – Weather Web Service URL
- **Optional** – false
- **Type** – string
- **Tooltip** – The URL to the Weather Web Service

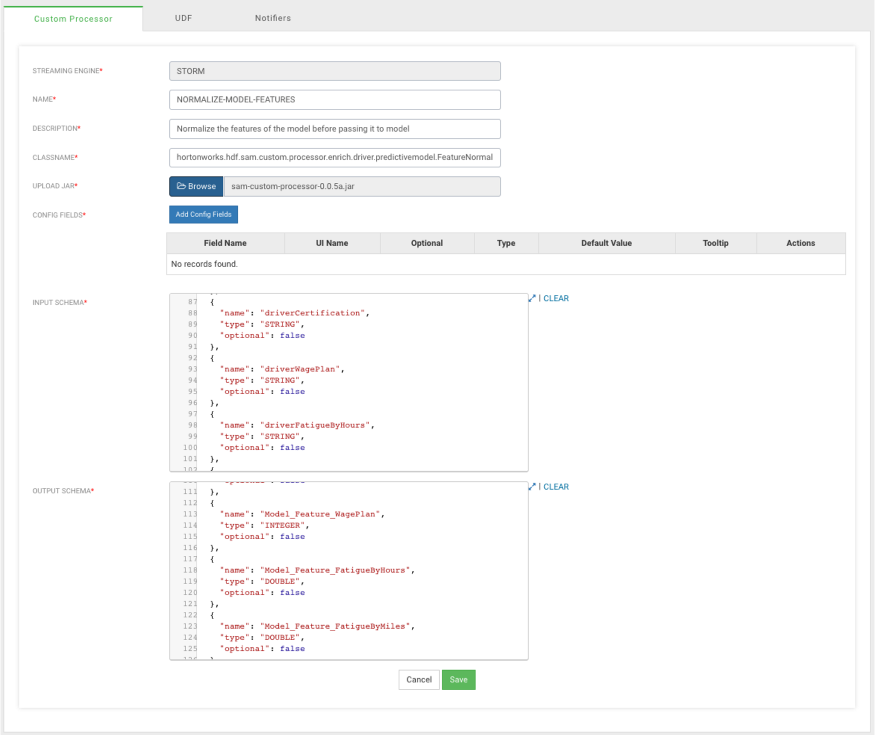
Input and Output Schema for ENRICH-WEATHER

- Copy this input schema and paste into the INPUT SCHEMA text area box
- Copy this output schema and paste into the OUTPUT SCHEMA text area box

Steps for Uploading the NORMALIZE-MODEL-FEATURES Custom Processor

1. Select the Custom Processor Tab and click the + sign to create the NORMALIZE-MODEL-FEATURES processor. This processor is to normalized the enriched fields to format that the model is expecting.

2. Configure the processor with the following values.



STREAMING ENGINE* STORM

NAME* NORMALIZE-MODEL-FEATURES

DESCRIPTION* Normalize the features of the model before passing it to model

CLASSNAME* hortonworks.hdf.sam.custom.processor.enrich.driver.predictivemodel.FeatureNormal

UPLOAD JAR* sam-custom-processor-0.0.5a.jar

CONFIG FIELDS*

Field Name	UI Name	Optional	Type	Default Value	Tooltip	Actions
No records found.						

INPUT SCHEMA*

```
{
  "name": "driverCertification",
  "type": "STRING",
  "optional": false
},
{
  "name": "driverWagePlan",
  "type": "STRING",
  "optional": false
},
{
  "name": "driverFatigueByHours",
  "type": "STRING",
  "optional": false
},
```

OUTPUT SCHEMA*

```
},
{
  "name": "Model_Feature_WagePlan",
  "type": "INTEGER",
  "optional": false
},
{
  "name": "Model_Feature_FatigueByHours",
  "type": "DOUBLE",
  "optional": false
},
{
  "name": "Model_Feature_FatigueByMiles",
  "type": "DOUBLE",
  "optional": false
```

NORMALIZE-MODEL-FEATURES Configuration Values

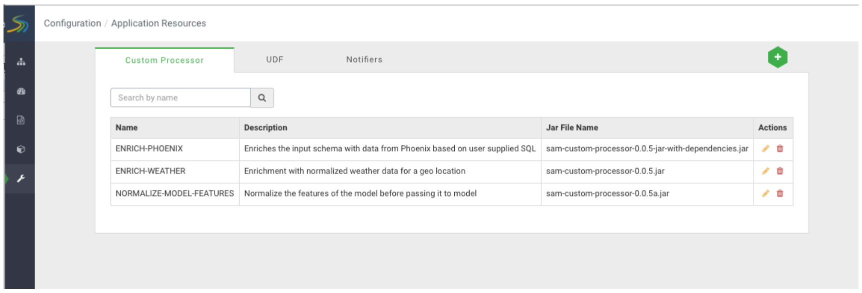
- **Streaming Engine** – Storm
- **Name** – NORMALIZE-MODEL-FEATURES
- **Description** – Normalize the features of the model before passing it to model
- **ClassName** – hortonworks.hdf.sam.custom.processor.enrich.driver.predictivemodel.FeatureNormalizationProcessor
- **Upload Jar** – The jar for this custom processor can be found under `SAM_EXTENSIONS/sam-custom-processor-0.0.5a.jar`

Input and Output Schema for NORMALIZE-MODEL-FEATURES

- Copy this input schema and paste into the INPUT SCHEMA text area box
- Copy this output schema and paste into the OUTPUT SCHEMA text area box

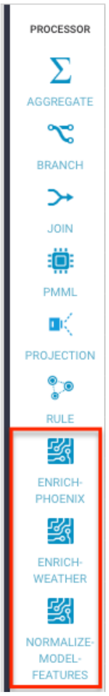
Result

We have uploaded three custom processors required to do enrichment of the stream and normalization of the enriched values to feed into the model.



Name	Description	Jar File Name	Actions
ENRICH-PHOENIX	Enriches the input schema with data from Phoenix based on user supplied SQL	sam-custom-processor-0.0.5-jar-with-dependencies.jar	✎ 🗑
ENRICH-WEATHER	Enrichment with normalized weather data for a geo location	sam-custom-processor-0.0.5.jar	✎ 🗑
NORMALIZE-MODEL-FEATURES	Normalize the features of the model before passing it to model	sam-custom-processor-0.0.5a.jar	✎ 🗑

If you go back to the Stream Builder, you will see three new custom processors on palette.



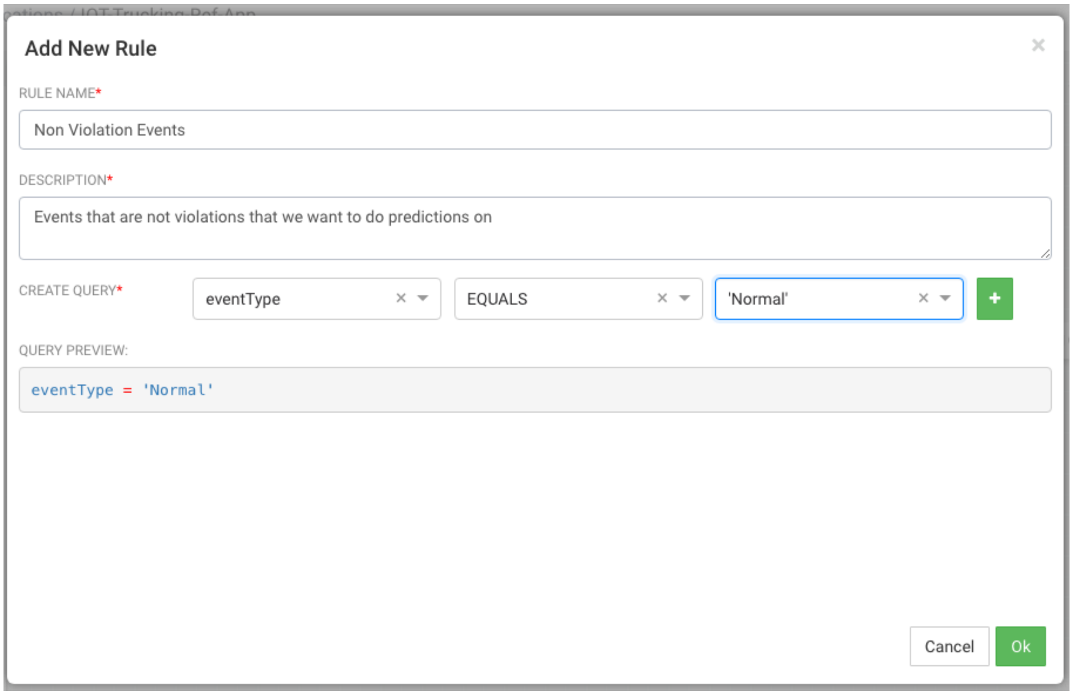
7.6. Scoring the Model in the Stream using a Streaming Split Join Pattern

About This Task

Now that you have created the enrichment store, loaded the enrichment data and uploaded the custom UDFs and processors to SAM, build the stream flow to score the model in real-time. In this case, you want to predict violations for events that are not blatant infractions.

Steps

1. Click into to the Trucking IOT application you built.
2. Double click the Event Type rule processor to display the **Add New Rule** dialog.
3. Configure the new rule with the following values:



Add New Rule

RULE NAME*

Non Violation Events

DESCRIPTION*

Events that are not violations that we want to do predictions on

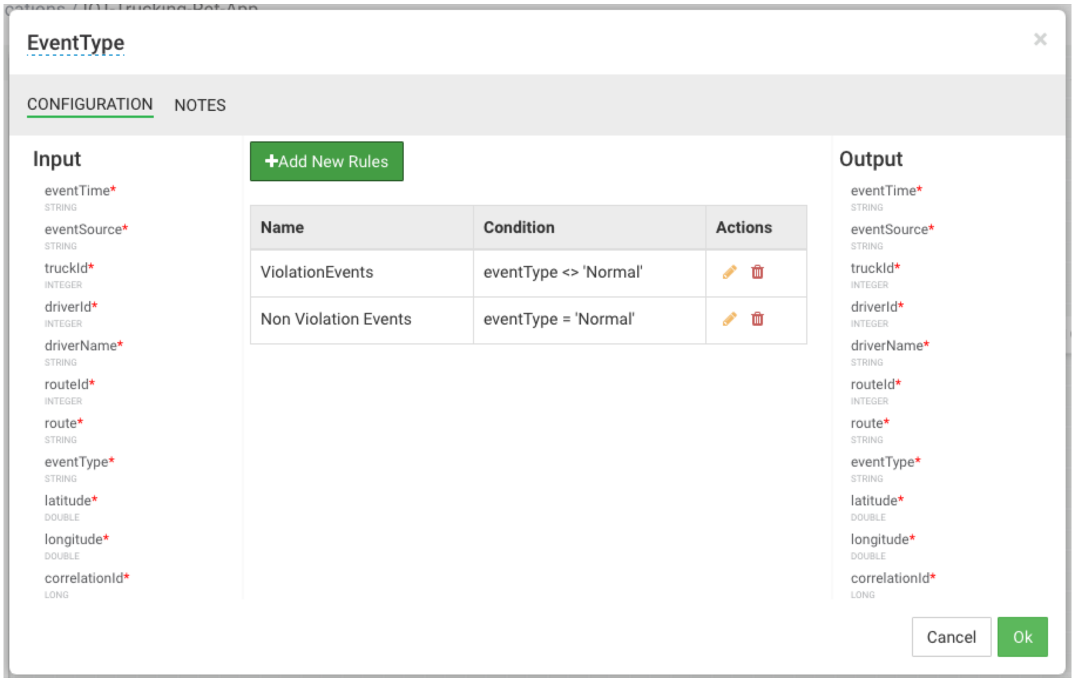
CREATE QUERY*: eventType EQUALS 'Normal'

QUERY PREVIEW:

```
eventType = 'Normal'
```

Result

Your new rule is added to the Event Type processor.



EventType

Input

- eventTime* STRING
- eventSource* STRING
- truckId* INTEGER
- driverId* INTEGER
- driverName* STRING
- routeId* INTEGER
- route* STRING
- eventType* STRING
- latitude* DOUBLE
- longitude* DOUBLE
- correlationId* LONG

Name	Condition	Actions
ViolationEvents	eventType <> 'Normal'	✎ 🗑
Non Violation Events	eventType = 'Normal'	✎ 🗑

Output

- eventTime* STRING
- eventSource* STRING
- truckId* INTEGER
- driverId* INTEGER
- driverName* STRING
- routeId* INTEGER
- route* STRING
- eventType* STRING
- latitude* DOUBLE
- longitude* DOUBLE
- correlationId* LONG

7.7. Streaming Split Join Pattern

About This Task

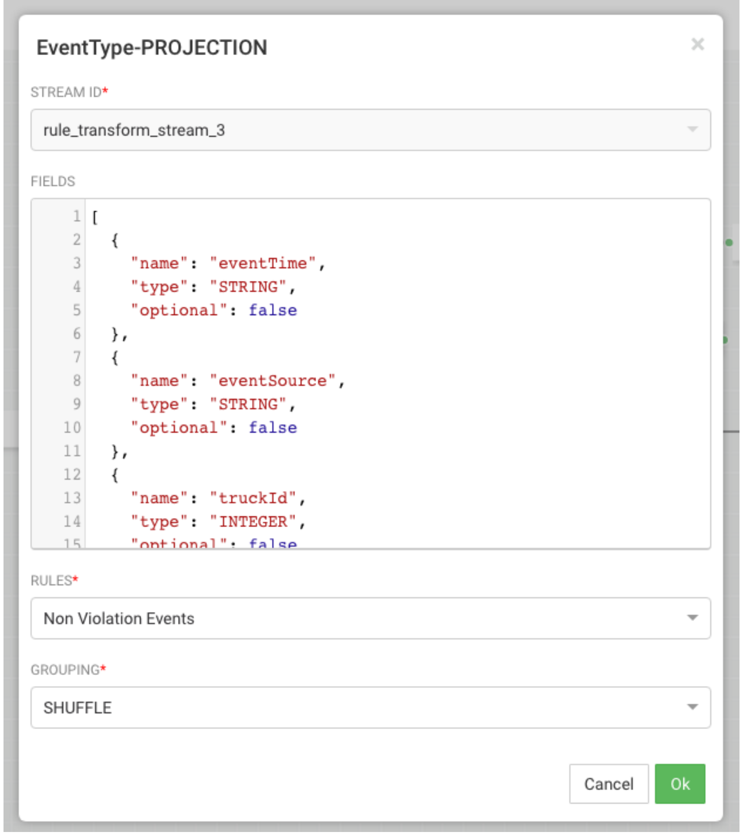
You objective is to perform three enrichments:

- Retrieve a driver's certification and wage plan from the driver's table
- Retrieve the driver's hours and miles logged from the timesheet table
- Query weather information for a specific time and location.

To do this, use the split join pattern to split the stream into 3, perform the enrichment in parallel, and then re-join the three streams.

Steps for Creating a Split Join Key

1. Create a new split key in the stream which allows you to join in a common field when you join the three stream. To do this, drag the projection processor to the canvas and create a connection from the EventType rule processor to this projectioin processor. When configuring the connection, select the Non Violation Events Rule which tells SAM to only send non violation events to this project processor.



EventType-PROJECTION

STREAM ID*

rule_transform_stream_3

FIELDS

```
[
  {
    "name": "eventTime",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "eventSource",
    "type": "STRING",
    "optional": false
  },
  {
    "name": "truckId",
    "type": "INTEGER",
    "optional": false
```

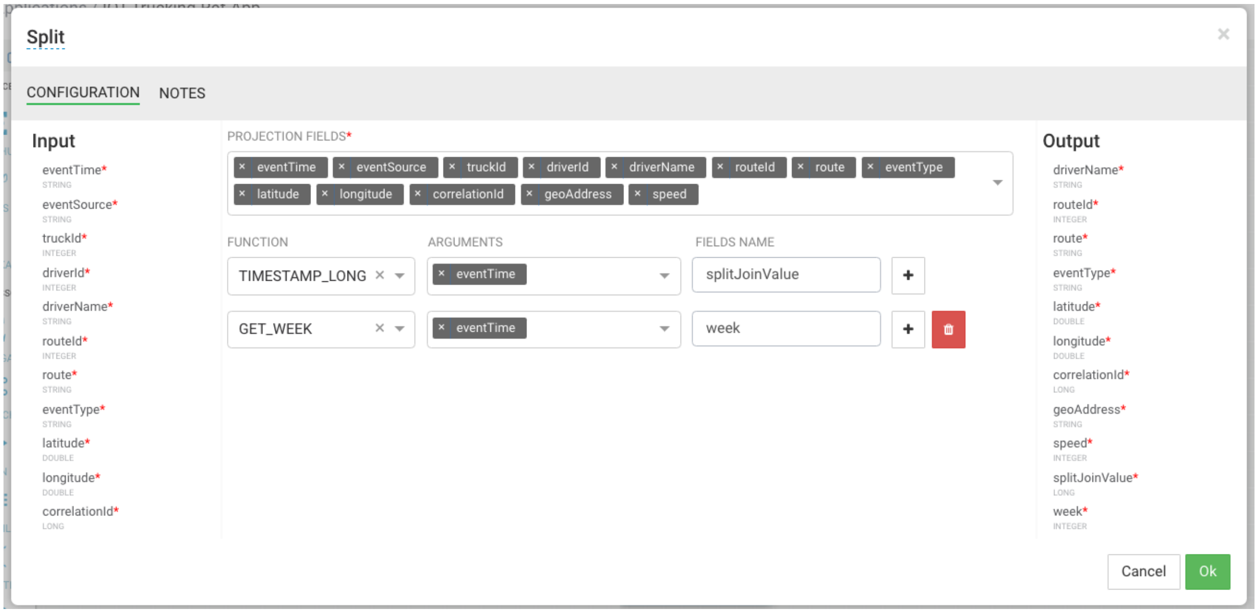
RULES*

Non Violation Events

GROUPING*

SHUFFLE

2. Configure the projection processor to create our split join key called splitJoinValue using the custom udf we uploaded earlier called "TIMESTAMP_LONG". We will also do a transformation which calculates the week based on the event time which is required for one of the enrichments downstream. Configure the processor like the following:



Split

CONFIGURATION NOTES

Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

PROJECTION FIELDS*

eventTime, eventSource, truckId, driverId, driverName, routeId, route, eventType, latitude, longitude, correlationId, geoAddress, speed

FUNCTION	ARGUMENTS	FIELDS NAME
TIMESTAMP_LONG	eventTime	splitJoinValue
GET_WEEK	eventTime	week

Output

driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG
geoAddress* STRING
speed* INTEGER
splitJoinValue* LONG
week* INTEGER

Cancel Ok

Steps for Splitting the Stream into Three to Perform Enrichments in Parallel

1. With the split join key created, we can split the stream into three to perform the enrichments in parallel. To do the first split to do the enrichment of the wage and certification status of driver, drag the "ENRICH-PHOENIX" processor the canvas and connect it from the Split project processor

2. Configure the enrich processor like below. After this processor executes, the output schema will have two fields populated called driverCertification and driverWagePlan.

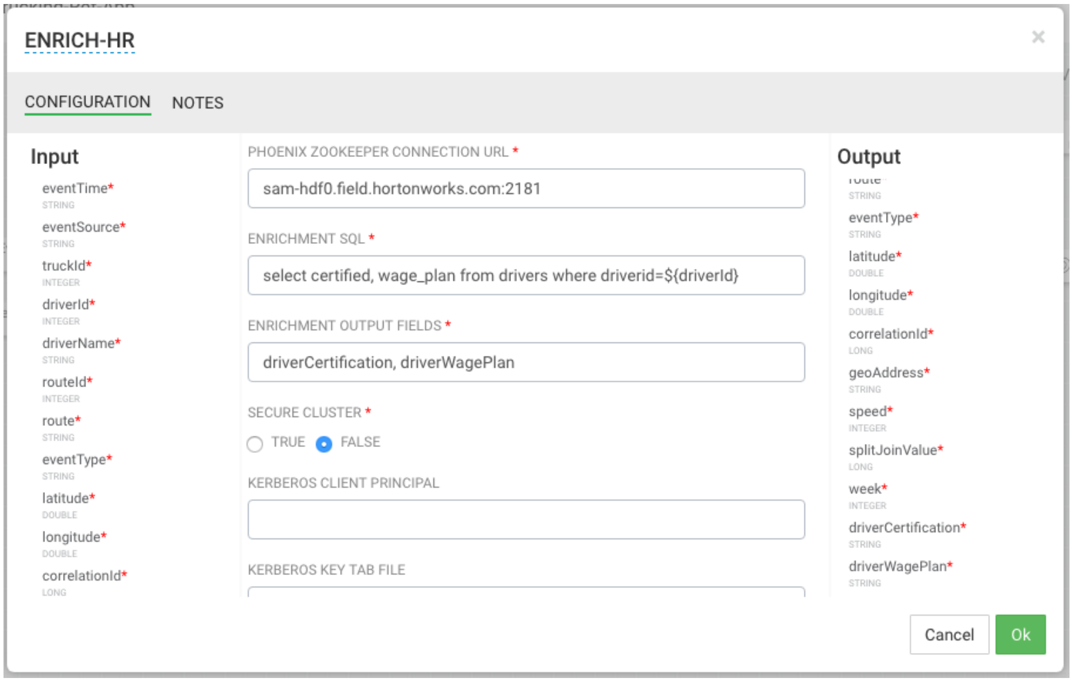
   a. ENRICHEMNT SQL: select certified, wage_plan from drivers where driverid= ${driverId}

   b. ENRICHMENT OUTPUT FIELDS: driverCertification, driverWagePlan

   c. SECURE CLUSTER: false

   d. INPUT SCHEMA MAPPINGS: Leave defaults

   e. OUTPUT FIELDS: select all fields except for driverFatigueByHours and driverFatigueByMiles



ENRICH-HR

CONFIGURATION NOTES

Input

- eventTime* STRING
- eventSource* STRING
- truckId* INTEGER
- driverId* INTEGER
- driverName* STRING
- routeId* INTEGER
- route* STRING
- eventType* STRING
- latitude* DOUBLE
- longitude* DOUBLE
- correlationId* LONG

PHOENIX ZOOKEEPER CONNECTION URL *

sam-hdf0.field.hortonworks.com:2181

ENRICHMENT SQL *

select certified, wage_plan from drivers where driverid=${driverId}

ENRICHMENT OUTPUT FIELDS *

driverCertification, driverWagePlan

SECURE CLUSTER *

○ TRUE ● FALSE

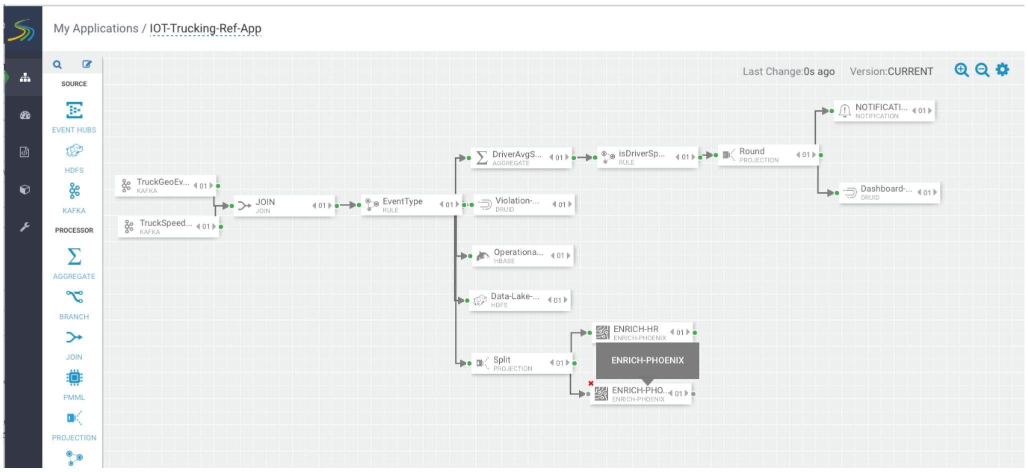
KERBEROS CLIENT PRINCIPAL

KERBEROS KEY TAB FILE

Output

- route STRING
- eventType* STRING
- latitude* DOUBLE
- longitude* DOUBLE
- correlationId* LONG
- geoAddress* STRING
- speed* INTEGER
- splitJoinValue* LONG
- week* INTEGER
- driverCertification* STRING
- driverWagePlan* STRING

3. Create the second stream to do enrichment of the drivers hours and miles logged in last week by dragging another "ENRICH-PHOENIX" processor to the canvas and connect it from the Split projection processor.



4. Configure the enrich processor like below. After this processor executes, the output schema will have two fields populated called driverFatigueByHours, driverFatigueByMiles.

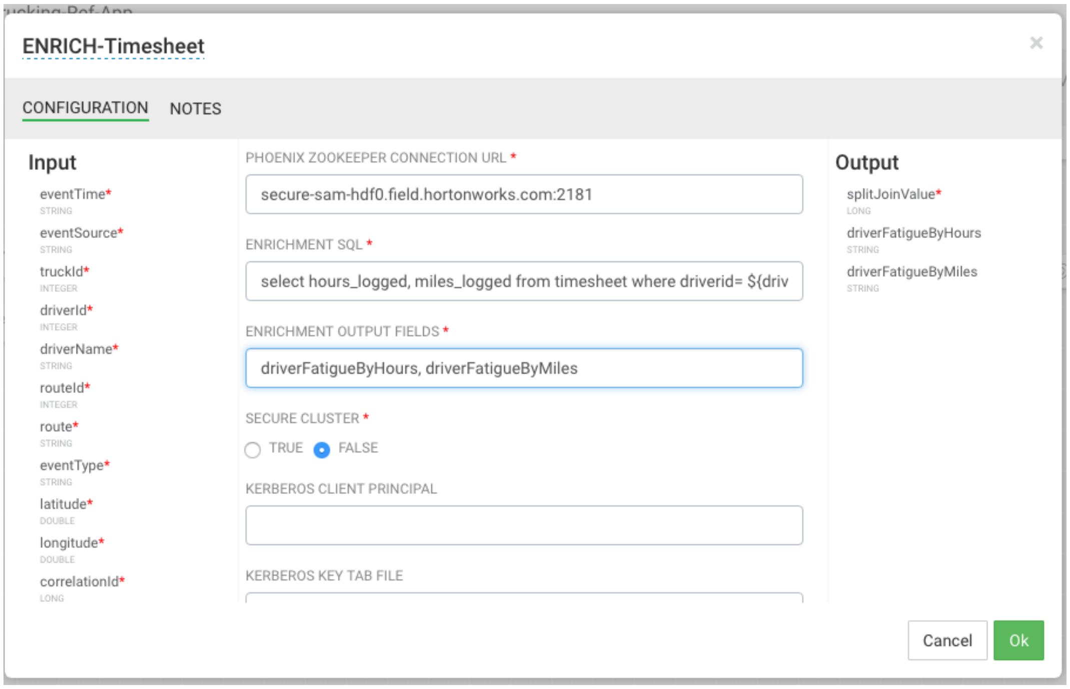
   a. ENRICHEMNT SQL: select hours_logged, miles_logged from timesheet where driverid=${driverId} and week=${week}

   b. ENRICHMENT OUTPUT FIELDS: driverFatigueByHours, driverFatigueByMiles

   c. SECURE CLUSTER: false

d. INPUT SCHEMA MAPPINGS: Leave defaults

e. OUTPUT FIELDS: select splitJoinValue, driverFatigueByHours, driverFatigueMiles



ENRICH-Timesheet

CONFIGURATION NOTES

Input

Field	Type
eventTime*	STRING
eventSource*	STRING
truckId*	INTEGER
driverId*	INTEGER
driverName*	STRING
routeId*	INTEGER
route*	STRING
eventType*	STRING
latitude*	DOUBLE
longitude*	DOUBLE
correlationId*	LONG

PHOENIX ZOOKEEPER CONNECTION URL *
secure-sam-hdf0.field.hortonworks.com:2181

ENRICHMENT SQL *
select hours_logged, miles_logged from timesheet where driverid= ${driv

ENRICHMENT OUTPUT FIELDS *
driverFatigueByHours, driverFatigueByMiles

SECURE CLUSTER *
○ TRUE ● FALSE

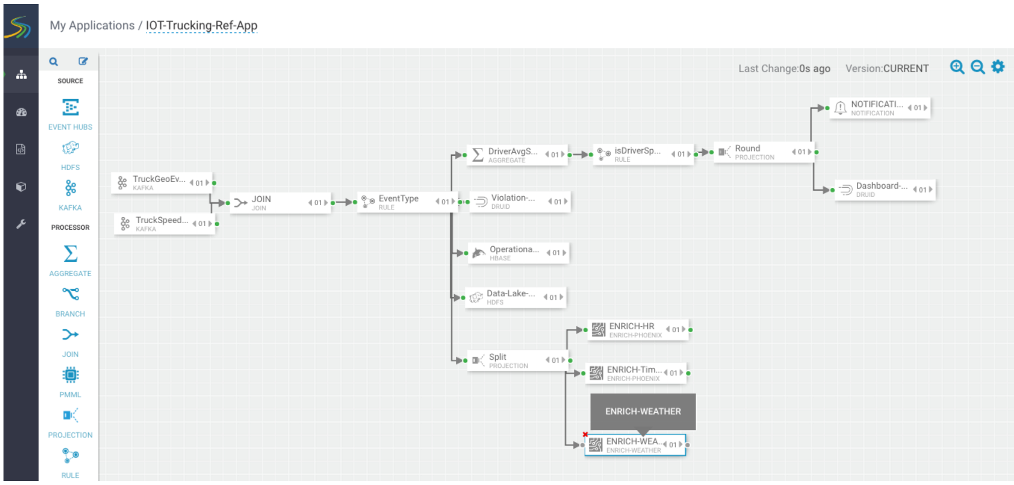
KERBEROS CLIENT PRINCIPAL

KERBEROS KEY TAB FILE

Output

Field	Type
splitJoinValue*	LONG
driverFatigueByHours	STRING
driverFatigueByMiles	STRING

5. Create the third stream to do weather enrichment by dragging the custom processor we uploaded called "ENRICH-WEATHER" processor to the canvas and connect it from the Split project processor.

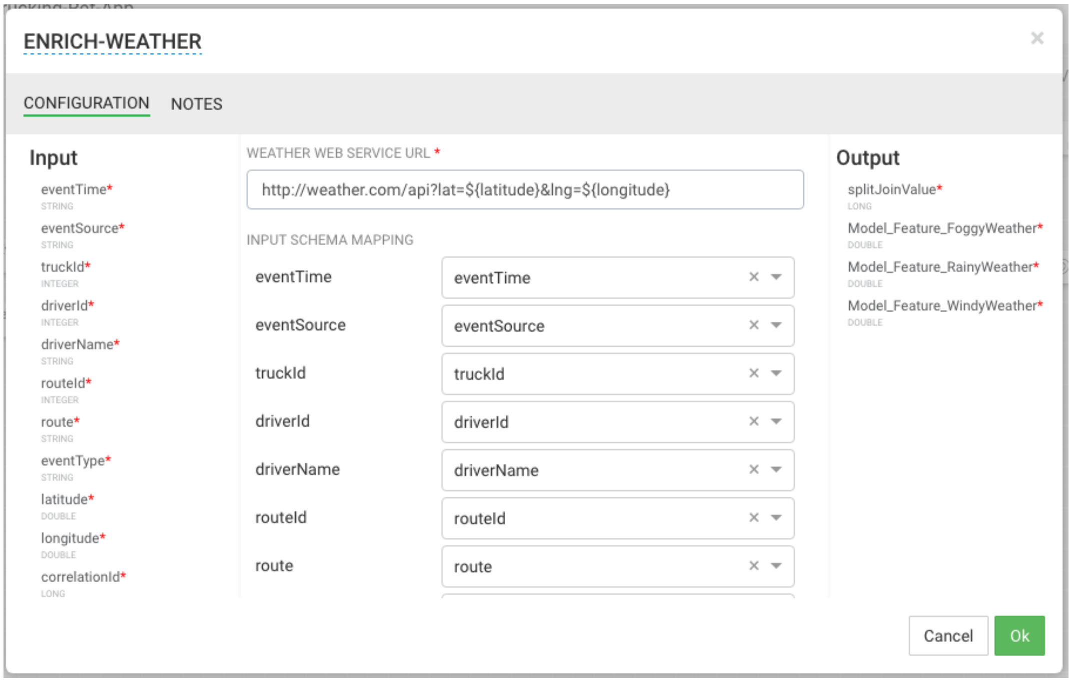


6. Configure the weather process like the following (currently the weather processor is just a stub that generates random normalized weather info). After this processor executes, the output schema will have three fields populated called Model_Feature_FoggyWeather, Model_Feature_RainyWeather, Model_Feature_WindyWeather.

a. WEATHER WEB SERVICE URL: http://weather.com/api?lat=${latitude}&lng=${longitude}

b. INPUT SCHEMA MAPPINGS: Leave defaults

c. OUTPUT FIELDS: Select the splitJoinValue and the three model enriched features



ENRICH-WEATHER

CONFIGURATION NOTES

Input

eventTime*	STRING
eventSource*	STRING
truckId*	INTEGER
driverId*	INTEGER
driverName*	STRING
routeId*	INTEGER
route*	STRING
eventType*	STRING
latitude*	DOUBLE
longitude*	DOUBLE
correlationId*	LONG

WEATHER WEB SERVICE URL *

http://weather.com/api?lat=${latitude}&lng=${longitude}

INPUT SCHEMA MAPPING

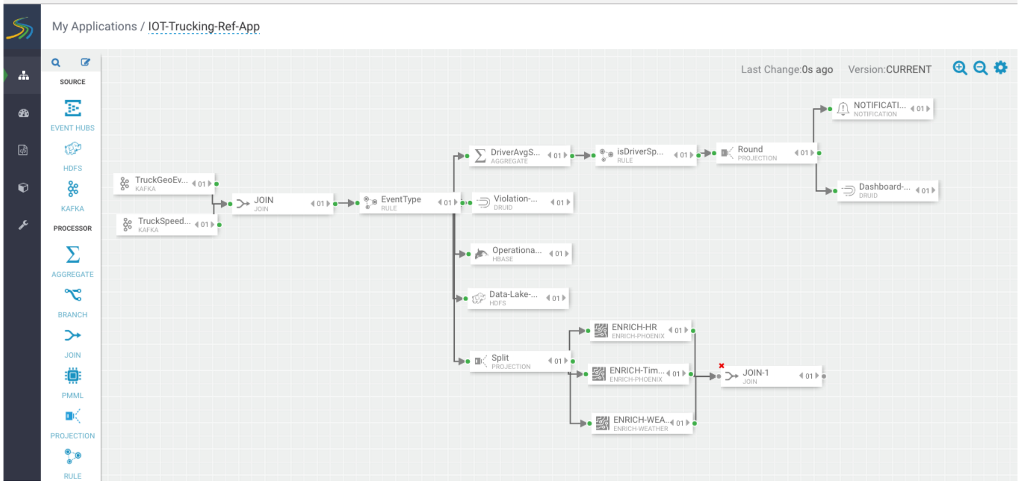
eventTime	eventTime
eventSource	eventSource
truckId	truckId
driverId	driverId
driverName	driverName
routeId	routeId
route	route

Output

splitJoinValue*	LONG
Model_Feature_FoggyWeather*	DOUBLE
Model_Feature_RainyWeather*	DOUBLE
Model_Feature_WindyWeather*	DOUBLE

Steps for Rejoining the Three Enriched Streams

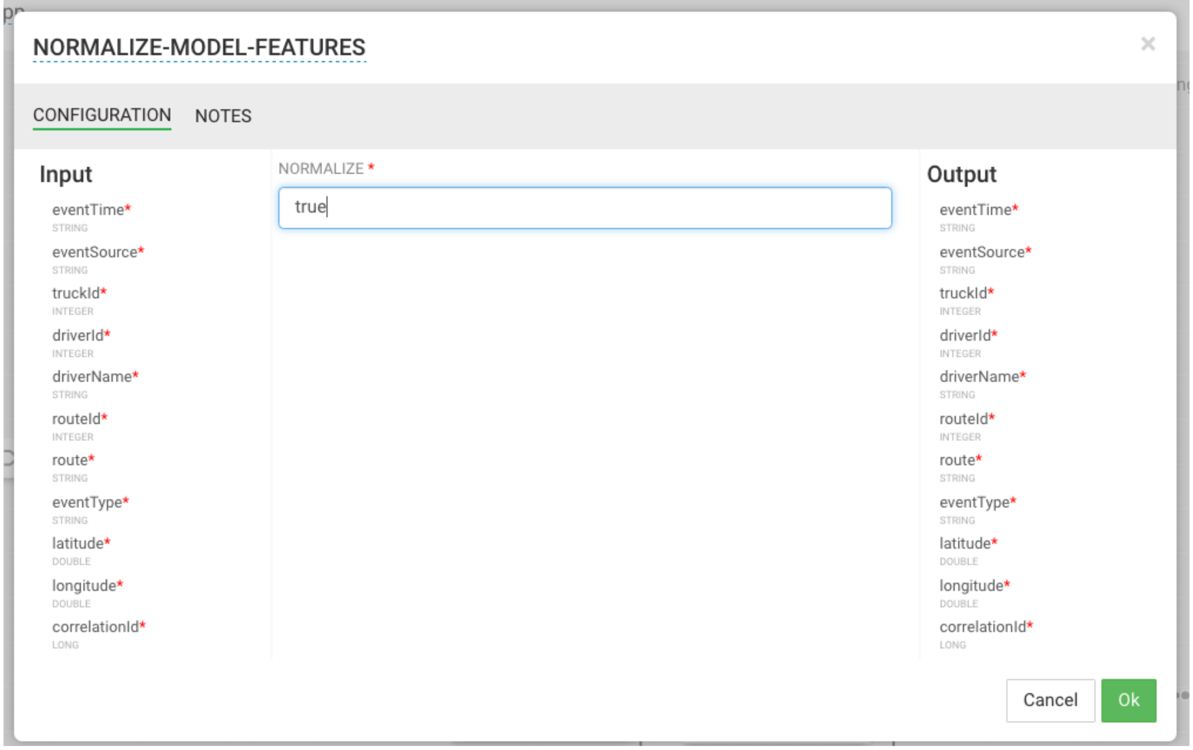
1. Now that we have done the enrichment in parallel by splitting the stream into 3, we can now join the 3 streams by dragging the join processor to the canvas and connecting the join from the 3 streams.



2. Configure the join processor like the following where we use the joinSplitValue to join all three streams.

For the Output field, just click **SELECT ALL** to get all the fields across the three streams.

3. Now that we have joined three enriched streams, lets normalize the data into the format that the model expects by dragging to the canvas the "NORMALIZE-MODEL-FEATURES" custom processor that we added. For the output fields select all the fields and the leave the the mapping as defaults.



NORMALIZE-MODEL-FEATURES

CONFIGURATION NOTES

Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

NORMALIZE *

true

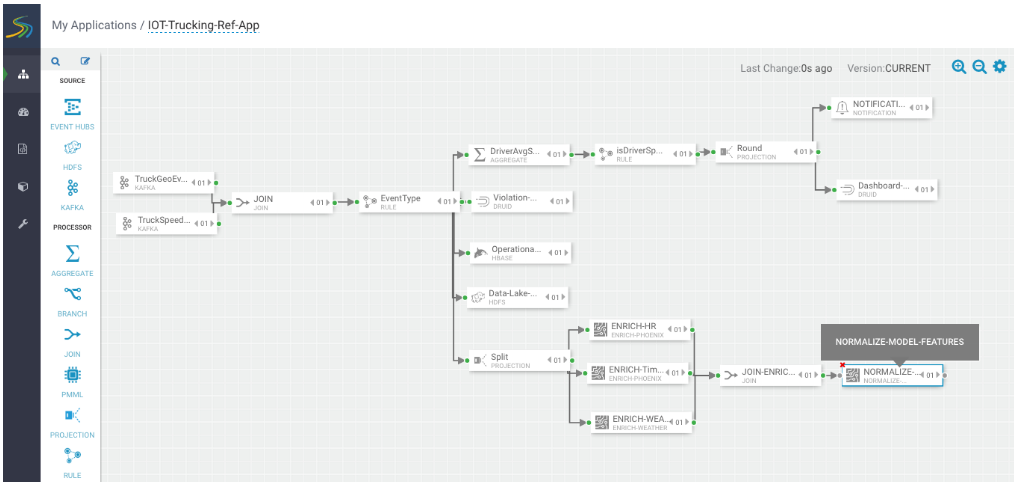
Output

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

Cancel Ok

Result

Your flow looks similar to the following.



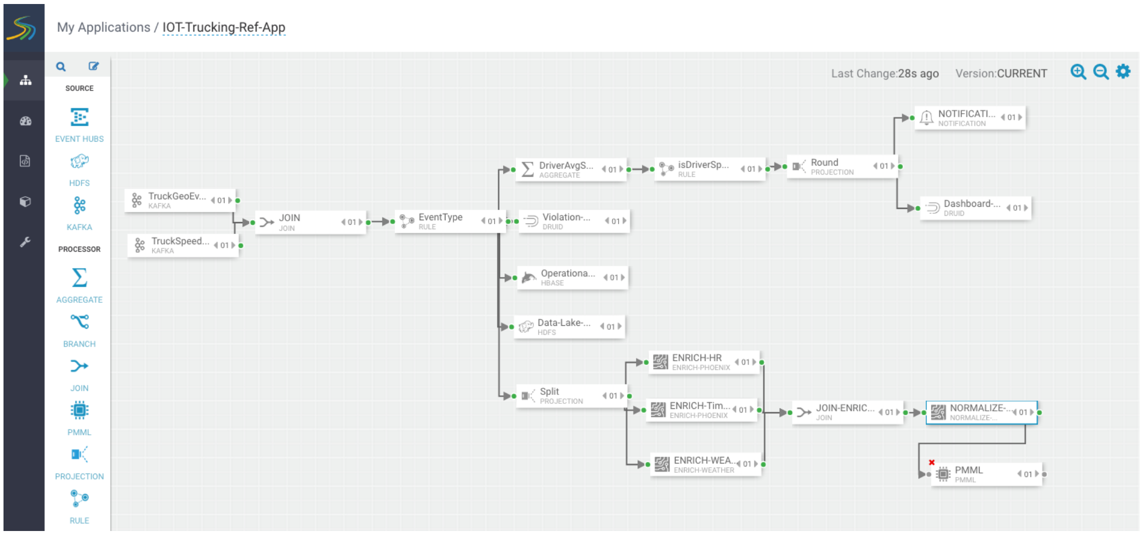
7.8. Score the Model using the PMML Processor and Alert

About This Task

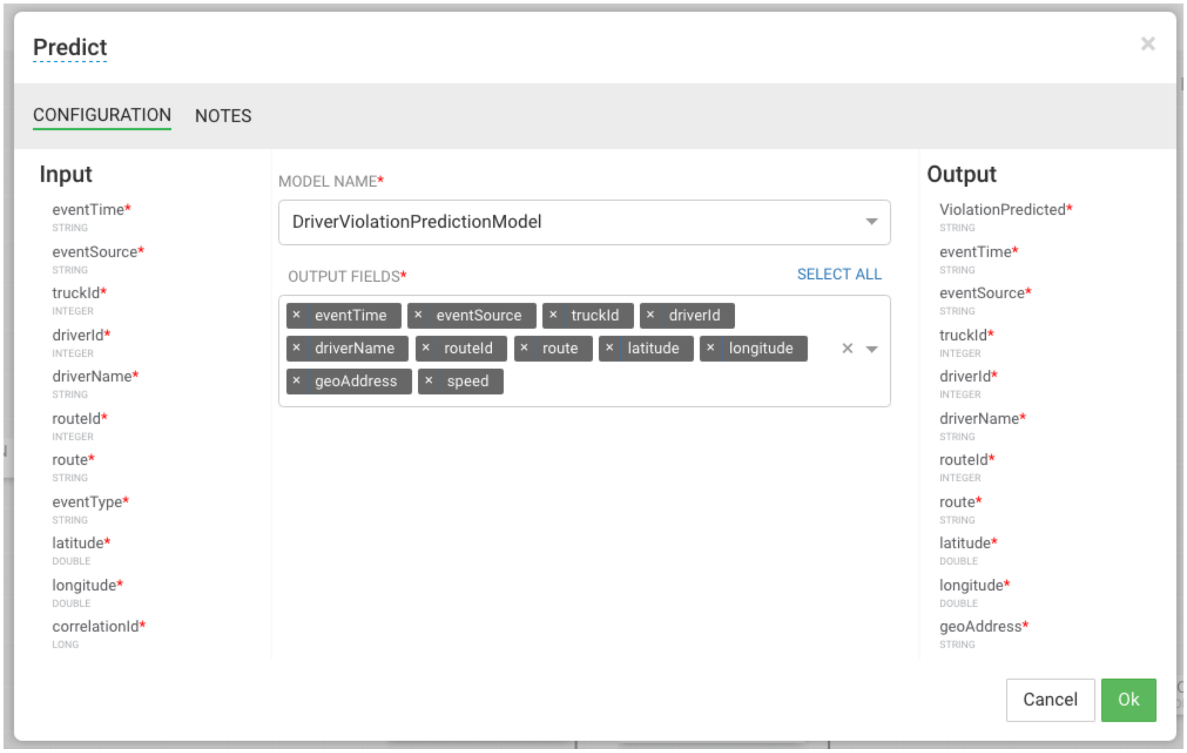
Now you are ready to score the logistical regression model.

Steps

1. Drag the PMML processor the canvas and connect it to the Normalize processor.



2. Configure the PMML processor like the following by selecting the DriverViolationPredictionModel that you uploaded to the **Model Registry** earlier. After this processor executes, a new field called **ViolationPredicted** is added to stream for the result of the prediction. In output fields, select all the contextual fields you want to pass on including the model value result.



Predict

Input

eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE
correlationId* LONG

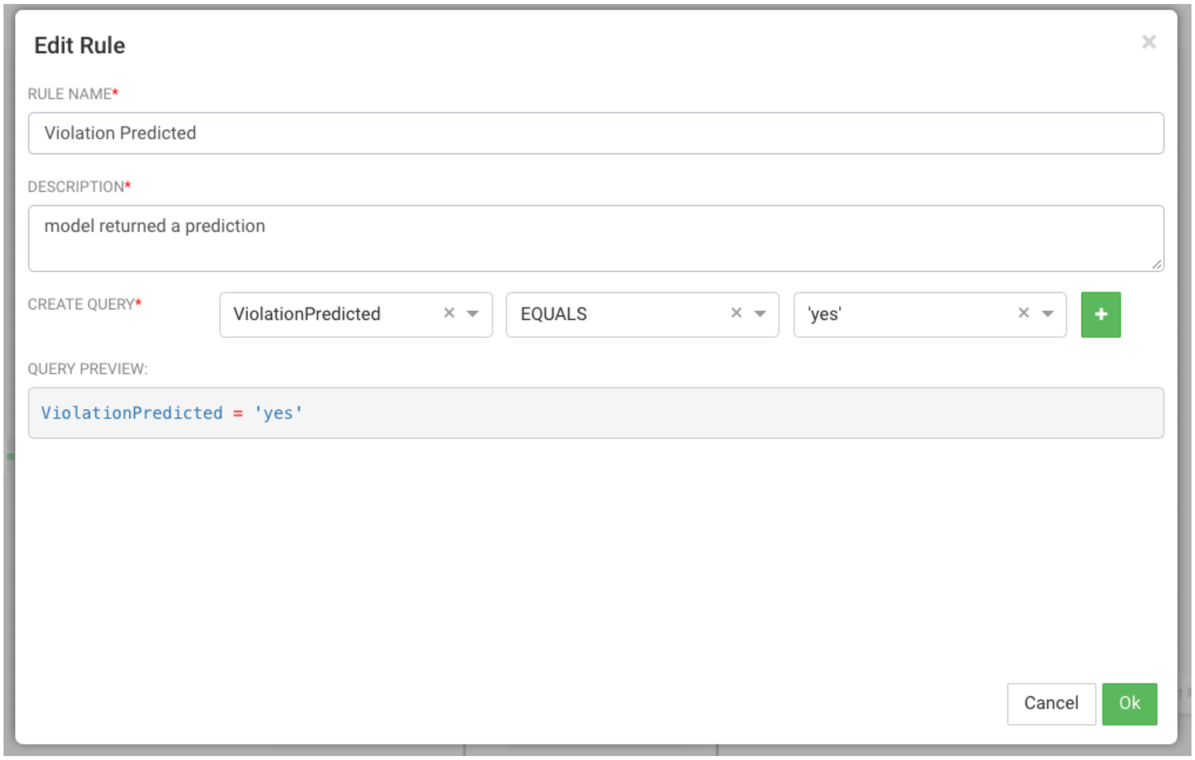
MODEL NAME*: DriverViolationPredictionModel

OUTPUT FIELDS*: eventTime, eventSource, truckId, driverId, driverName, routeId, route, latitude, longitude, geoAddress, speed

Output

ViolationPredicted* STRING
eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
latitude* DOUBLE
longitude* DOUBLE
geoAddress* STRING

3. Determine if the model predicted if the driver will commit a violation by dragging a rule processor to the canvas and configuring a rule like the following:



Edit Rule

RULE NAME*: Violation Predicted

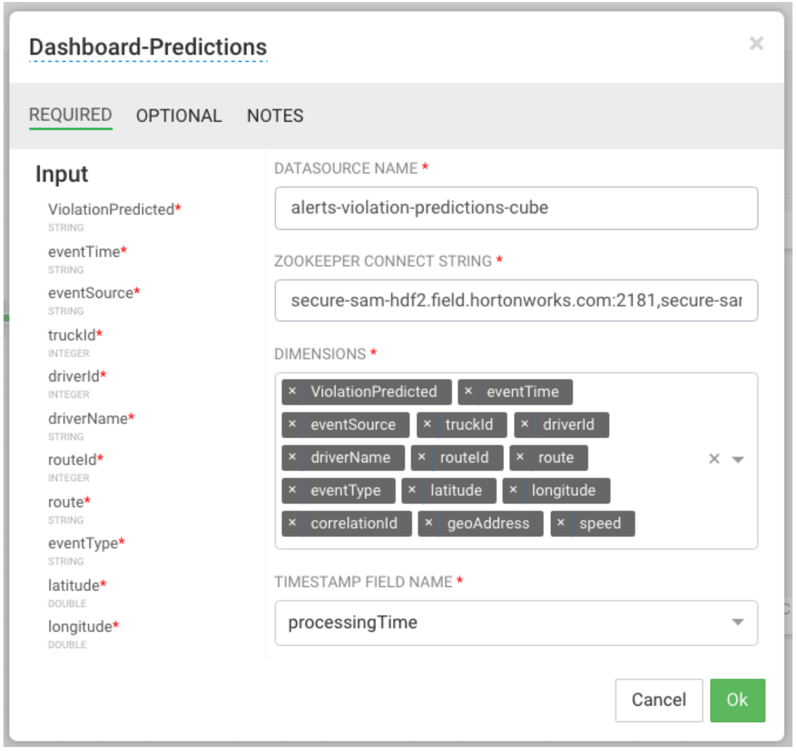
DESCRIPTION*: model returned a prediction

CREATE QUERY*: ViolationPredicted | EQUALS | 'yes'

QUERY PREVIEW:

```
ViolationPredicted = 'yes'
```

4. If a violation is predicted, send it to a Druid to display on a dashboard. Drag the Druid processor to canvas and configure. Stream the events into a cube called **alerts-violation-predictions-cube**.



Dashboard-Predictions

REQUIRED OPTIONAL NOTES

Input

ViolationPredicted* STRING
eventTime* STRING
eventSource* STRING
truckId* INTEGER
driverId* INTEGER
driverName* STRING
routeId* INTEGER
route* STRING
eventType* STRING
latitude* DOUBLE
longitude* DOUBLE

DATASOURCE NAME *
alerts-violation-predictions-cube

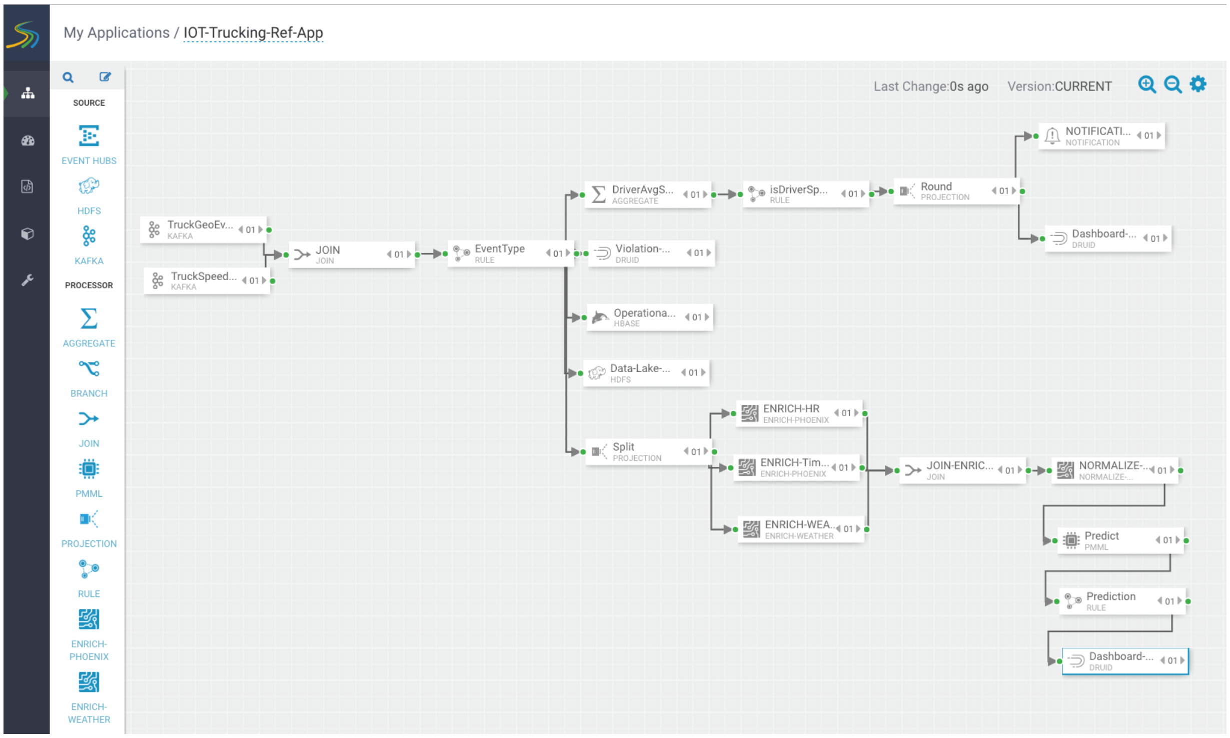
ZOOKEEPER CONNECT STRING *
secure-sam-hdf2.field.hortonworks.com:2181,secure-sai

DIMENSIONS *
× ViolationPredicted × eventTime × eventSource × truckId × driverId × driverName × routeId × route × eventType × latitude × longitude × correlationId × geoAddress × speed

TIMESTAMP FIELD NAME *
processingTime

Result

The final flow looks like the following:



My Applications / IOT-Trucking-Ref-App

8. Creating Visualizations Using Superset

A business analyst can create a wide array of visualizations to gather insights on streaming data. The platform supports over 30+ visualizations the business analyst can create. For visualization examples, see the Gallery of Superset Visualizations.

The general process for creating and viewing visualizations is as follows:

1. Whenever you add new data sources to Druid via a Stream App, perform the **Refresh Druid Metadata** action on the Superset menu.

2. Using the Superset Stream Insight UI, create one or more "slices". A slice is one business visualization assoicated with a data source (e.g: Druid cube).

3. Using the Dashboard menu, add the slices to your dashboard and organize their layout.

Note

Note that when a SAM app streams data to a new cube using the Druid processor, it will about 30 minutes for the cube to appear in Superset. This is because Superset has to wait for the first segment to be created in Druid. After the cube appears, users can analyze the streaming data immediately as it is streaming in.

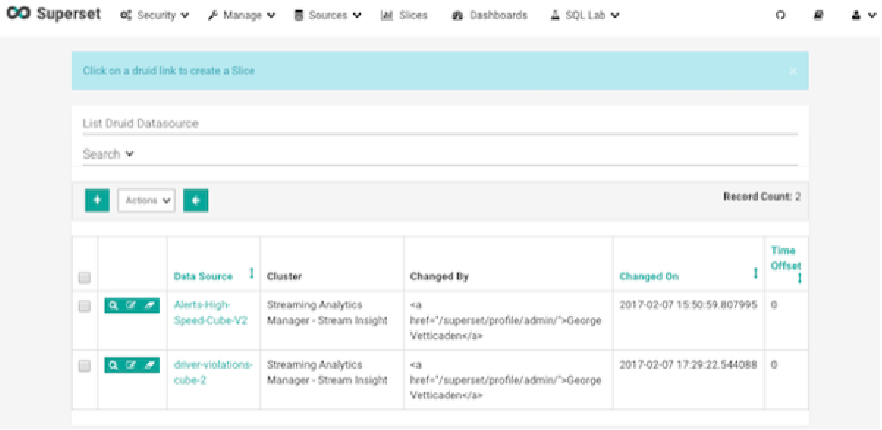
8.1. Creating Insight Slices

The following steps demonstrate a typical flow for creating a slice:

1. Choose **Slices** on the Menu.

2. Click + to create a new Slice.

3. Select the Druid Data Source that you want to use for the new visualization:



Click on a druid link to create a Slice

List Druid Datasource

Search

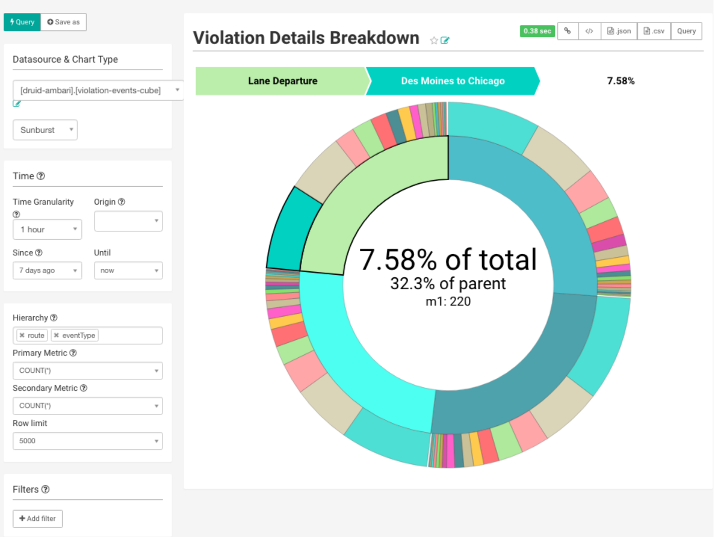
Actions

Record Count: 2

		Data Source	Cluster	Changed By	Changed On	Time Offset
□		Alerts-High-Speed-Cube-V2	Streaming Analytics Manager - Stream Insight	<a href="/superset/profile/admin/">George Vetticaden</a>	2017-02-07 15:50:59.807995	0
□		driver-violations-cube-2	Streaming Analytics Manager - Stream Insight	<a href="/superset/profile/admin/">George Vetticaden</a>	2017-02-07 17:29:22.544088	0

4. Select a Chart Type from the menu.

This example creates a "Sunburst" visualization where we are rolling up multiple dimensions like route, eventType and driver info. Configure the chart and click **Execute Query**.



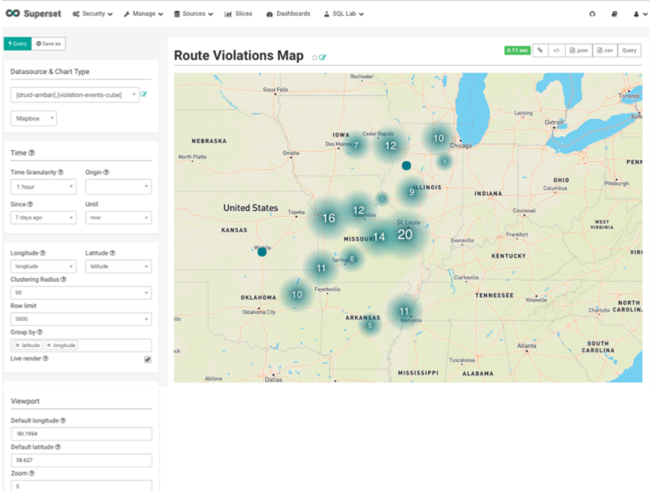
Datasource & Chart Type
[druid-ambari].[violation-events-cube]
Sunburst

Time
Time Granularity: 1 hour
Origin:
Since: 7 days ago
Until: now

Hierarchy: route, eventType
Primary Metric: COUNT(*)
Secondary Metric: COUNT(*)
Row limit: 5000

Filters
Add filter

5. Another visualization could be integration with MapBox Here we are mapping where violations are occurring the most based on the lat/long location of the event



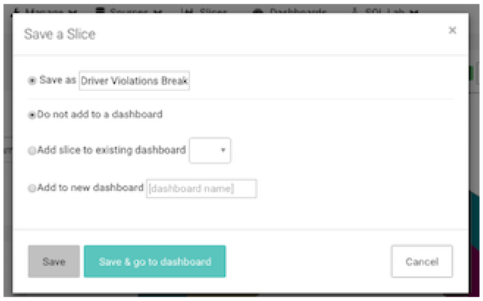
Datasource & Chart Type
[druid-ambari].[violation-events-cube]
Mapbox

Time
Time Granularity: 1 hour
Origin:
Since: 7 days ago
Until: now

Longitude: longitude
Latitude: latitude
Clustering Radius: 60
Row limit: 5000
Group by: latitude, longitude
Live render ☑

Viewport
Default longitude: -90.1994
Default latitude: 38.627
Zoom: 5

6. To save the slice, specify a name and name and click **Save**.



Save a Slice ×

● Save as Driver Violations Break

● Do not add to a dashboard

○ Add slice to existing dashboard

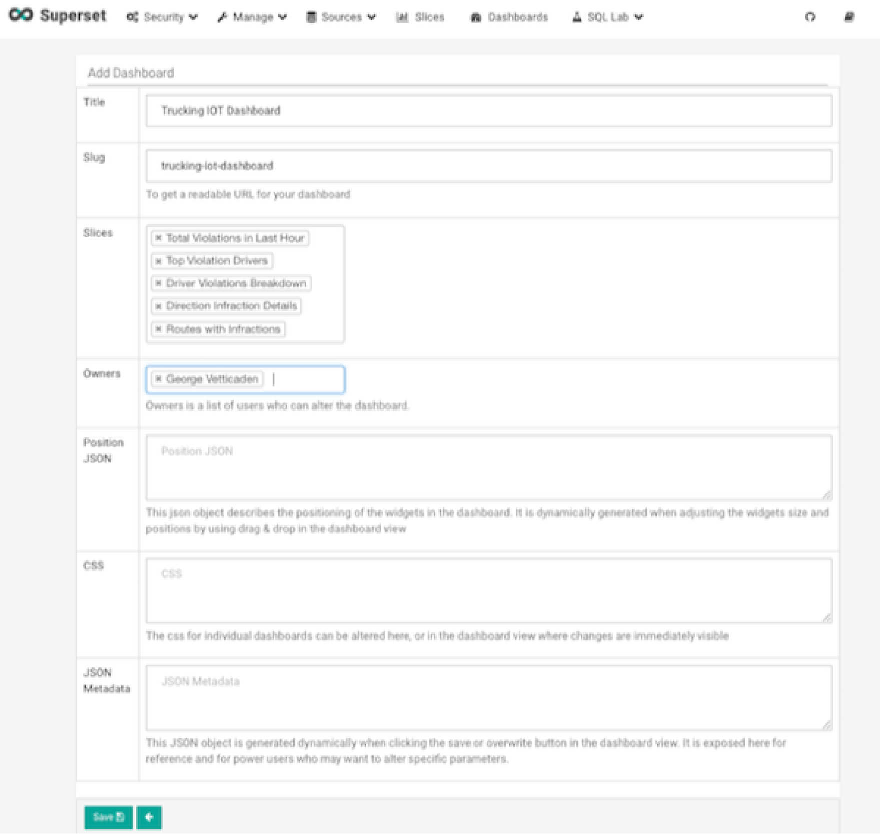
○ Add to new dashboard [dashboard name]

Save | Save & go to dashboard | Cancel

8.2. Adding Insight Slices to a Dashboard

After you create slices, you can organize them into a dashboards:

1. Click the Dashboard menu item.
2. Click + to create a new Dashboard.
3. Configure the dashboard: specify a name and the slices to include in the Dashboard.



Add Dashboard

Title: Trucking IOT Dashboard

Slug: trucking-iot-dashboard

To get a readable URL for your dashboard

Slices: × Total Violations in Last Hour, × Top Violation Drivers, × Driver Violations Breakdown, × Direction Infraction Details, × Routes with Infractions

Owners: × George Vetticaden

Owners is a list of users who can alter the dashboard.

Position JSON: Position JSON

This json object describes the positioning of the widgets in the dashboard. It is dynamically generated when adjusting the widgets size and positions by using drag & drop in the dashboard view

CSS: CSS

The css for individual dashboards can be altered here, or in the dashboard view where changes are immediately visible

JSON Metadata: JSON Metadata

This JSON object is generated dynamically when clicking the save or overwrite button in the dashboard view. It is exposed here for reference and for power users who may want to alter specific parameters.

4. Arrange the slices on the dashboard as desired, and then click **Save**.

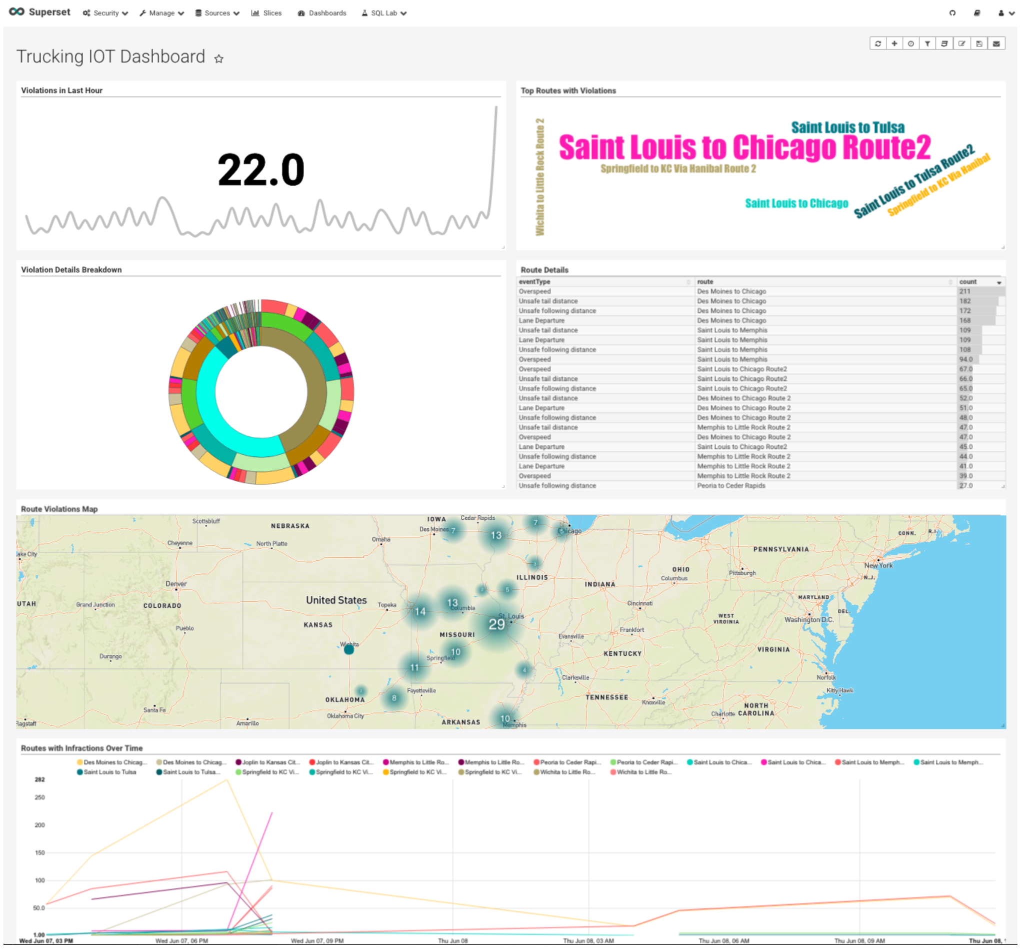
8.2.1. Dashboards for the Trucking IOT App

The IOT Trucking app that we implementing using the Stream Builder was streaming violation events, alerts and predictions into three cubes:

- violation-events-cube
- alerts-speeding-drivers-cube
- alerts-violation-predictions-cube

Based on the powerful visualizations that SuperSet offers, you can create the below powerful dashboards in minutes.

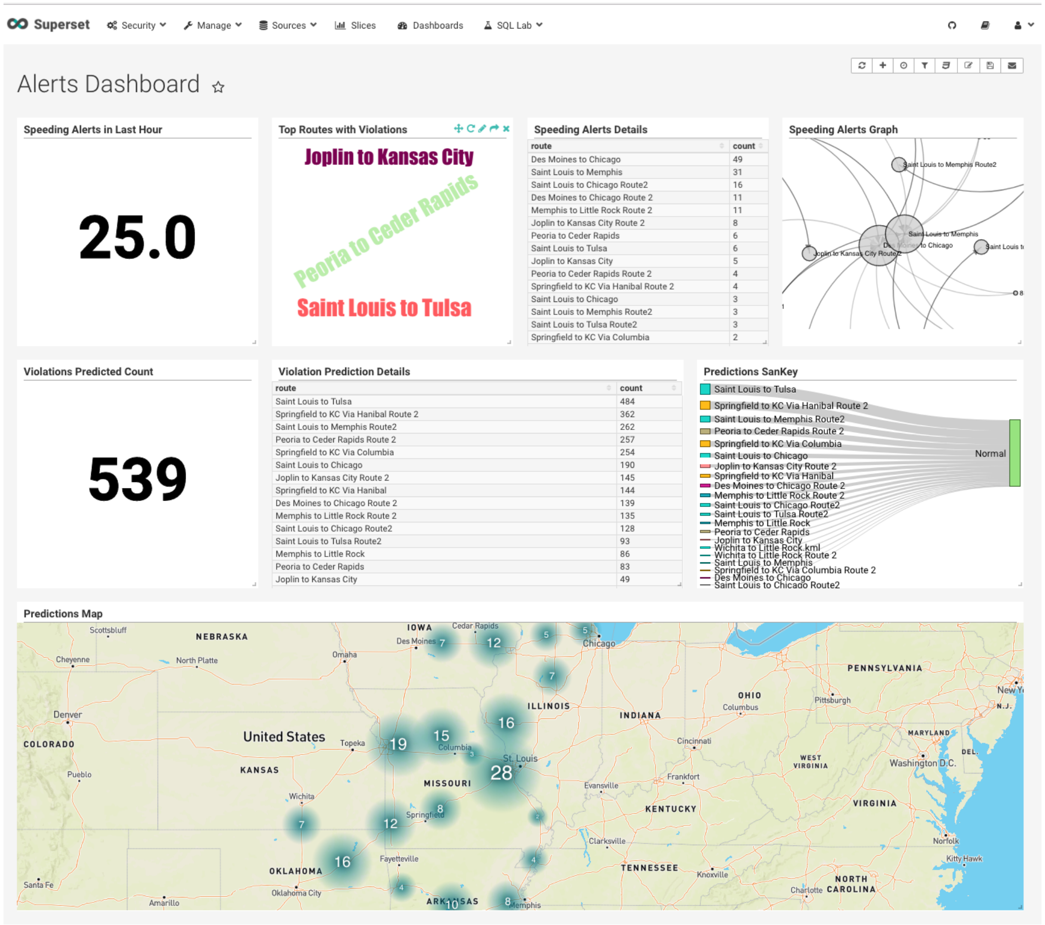
IoT Dashboard



Route Details

eventType	route	count
Overspeed	Des Moines to Chicago	211
Unsafe tail distance	Des Moines to Chicago	182
Unsafe following distance	Des Moines to Chicago	172
Lane Departure	Des Moines to Chicago	168
Unsafe tail distance	Saint Louis to Memphis	109
Lane Departure	Saint Louis to Memphis	109
Unsafe following distance	Saint Louis to Memphis	108
Overspeed	Saint Louis to Memphis	94.0
Overspeed	Saint Louis to Chicago Route2	67.0
Unsafe tail distance	Saint Louis to Chicago Route2	66.0
Unsafe following distance	Saint Louis to Chicago Route2	65.0
Unsafe tail distance	Des Moines to Chicago Route 2	52.0
Lane Departure	Des Moines to Chicago Route 2	51.0
Unsafe following distance	Des Moines to Chicago Route 2	48.0
Unsafe tail distance	Memphis to Little Rock Route 2	47.0
Overspeed	Des Moines to Chicago Route 2	47.0
Lane Departure	Saint Louis to Chicago Route2	45.0
Unsafe following distance	Memphis to Little Rock Route 2	44.0
Lane Departure	Memphis to Little Rock Route 2	41.0
Overspeed	Memphis to Little Rock Route 2	39.0
Unsafe following distance	Peoria to Ceder Rapids	27.0

Alerts Dashboard



Alerts Dashboard ☆

Speeding Alerts in Last Hour

25.0

Top Routes with Violations

Speeding Alerts Details

route	count
Des Moines to Chicago	49
Saint Louis to Memphis	31
Saint Louis to Chicago Route2	16
Des Moines to Chicago Route 2	11
Memphis to Little Rock Route 2	11
Joplin to Kansas City Route 2	8
Peoria to Ceder Rapids	6
Saint Louis to Tulsa	6
Joplin to Kansas City	5
Peoria to Ceder Rapids Route 2	4
Springfield to KC Via Hanibal Route 2	4
Saint Louis to Chicago	3
Saint Louis to Memphis Route2	3
Saint Louis to Tulsa Route2	3
Springfield to KC Via Columbia	2

Speeding Alerts Graph

Violations Predicted Count

539

Violation Prediction Details

route	count
Saint Louis to Tulsa	484
Springfield to KC Via Hanibal Route 2	362
Saint Louis to Memphis Route2	262
Peoria to Ceder Rapids Route 2	257
Springfield to KC Via Columbia	254
Saint Louis to Chicago	190
Joplin to Kansas City Route 2	145
Springfield to KC Via Hanibal	144
Des Moines to Chicago Route 2	139
Memphis to Little Rock Route 2	135
Saint Louis to Chicago Route2	128
Saint Louis to Tulsa Route2	93
Memphis to Little Rock	86
Peoria to Ceder Rapids	83
Joplin to Kansas City	49

Predictions SanKey

- Saint Louis to Tulsa
- Springfield to KC Via Hanibal Route 2
- Saint Louis to Memphis Route2
- Peoria to Ceder Rapids Route 2
- Springfield to KC Via Columbia
- Saint Louis to Chicago
- Joplin to Kansas City Route 2
- Springfield to KC Via Hanibal
- Des Moines to Chicago Route 2
- Memphis to Little Rock Route 2
- Saint Louis to Chicago Route2
- Saint Louis to Tulsa Route2
- Memphis to Little Rock
- Peoria to Ceder Rapids
- Joplin to Kansas City
- Wichita to Little Rock.kml
- Wichita to Little Rock Route 2
- Saint Louis to Memphis
- Springfield to KC Via Columbia Route 2
- Des Moines to Chicago
- Saint Louis to Chicago Route2

Predictions Map