Hortonworks DataFlow

Getting Started with Streaming Analytics

(June 6, 2018)

docs.cloudera.com

Hortonworks DataFlow: Getting Started with Streaming Analytics

Copyright © 2012-2018 Hortonworks, Inc. Some rights reserved.

Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
http://creativecommons.org/licenses/by-sa/4.0/legalcode

Table of Contents

1. Building an End-to-End Stream Application 1
    1.1. Understanding the Use Case 1
    1.2. Reference Architecture 2
2. Prepare Your Environment 4
    2.1. Deploying Your HDF Clusters 4
    2.2. Registering Schemas in Schema Registry 4
        2.2.1. Create the Kafka Topics 4
        2.2.2. Register Schemas for the Kafka Topics 5
    2.3. Setting up an Enrichment Store, Creating an HBase Table, and Creating an HDFS Directory 7
3. Creating a Dataflow Application 9
    3.1. Data Producer Application Generates Events 9
    3.2. NiFi: Create a Dataflow Application 10
        3.2.1. NiFi Controller Services 10
        3.2.2. NiFi Ingests the Raw Sensor Events 11
        3.2.3. Publish Enriched Events to Kafka for Consumption by Analytics Applications 13
        3.2.4. Start the NiFi Flow 15
4. Creating a Stream Analytics Application 16
    4.1. Two Options for Creating the Streaming Analytics Applications 16
        4.1.1. Setting up the Stream Analytics App using the TruckingRefAppEnvEnviornmentBuilder 17
        4.1.2. Configuring and Deploying the Reference Application 17
    4.2. Creating a Service Pool and Environment 18
    4.3. Creating Your First Application 18
    4.4. Creating and Configuring the Kafka Source Stream 19
    4.5. Connecting Components 21
    4.6. Joining Multiple Streams 21
    4.7. Filtering Events in a Stream using Rules 23
    4.8. Using Aggregate Functions over Windows 24
    4.9. Implementing Business Rules on the Stream 25
    4.10. Transforming Data using a Projection Processor 27
    4.11. Streaming Alerts to an Analytics Engine for Dashboarding 29
    4.12. Streaming Violation Events to an Analytics Engine for Descriptive Analytics 30
    4.13. Streaming Violation Events into a Data Lake and Operational Data Store 32
5. Deploy an Application 35
    5.1. Configure Deployment Settings 35
    5.2. Deploy the App 35
    5.3. Running the Stream Simulator 37
6. Advanced: Performing Predictive Analytics on the Stream 39
    6.1. Logistical Regression Model 40
    6.2. Export the Model into SAM's Model Registry 41
    6.3. Enrichment and Normalization of Model Features 42
    6.4. Upload Custom Processors and UDFs for Enrichment and Normalization 43
        6.4.1. Upload Custom UDFs 43
        6.4.2. Upload Custom Processors 44
    6.5. Scoring the Model in the Stream using a Streaming Split Join Pattern 50

6.6. Streaming Split Join Pattern 51
6.7. Score the Model Using the PMML Processor and Alert 58
7. Creating Visualizations Using Superset 62
7.1. Creating Insight Slices 62
7.2. Adding Insight Slices to a Dashboard 63
7.2.1. Dashboards for the Trucking IOT App 64
8. SAM Test Mode 67
8.1. Four Test Cases using SAM's Test Mode 67
8.1.1. Test Case 1: Testing Normal Event with No Violation Prediction 67
8.1.2. Analyzing Test Case 1 Results 69
8.1.3. Test Case 2: Testing Normal Event with Yes Violation Prediction 72
8.1.4. Analyzing Test Case 2 Results 72
8.1.5. Test Case 3: Testing Violation Event 73
8.1.6. Analyzing Test Case 3 Results 73
8.1.7. Test Case 4: Testing Multiple-Speeding-Events 75
8.1.8. Analyzing Test Case 4 Results 75
8.1.9. Running SAM Test Cases as Junit Tests in CI Pipelines 77
9. Creating Custom Sources and Sinks 81
9.1. Cloud Use Case: Integration with AWS Kinesis and S3 81
9.2. Registering a Custom Source in SAM for AWS Kinesis 82
9.3. Registering a Custom Sink in SAM for AWS S3 84
9.4. Implementing the SAM App with Kinesis Source and S3 Sink 86
10. Stream Operations 88
10.1. My Applications View 88
10.2. Application Performance Monitoring 88
10.3. Exporting and Importing Stream Applications 89
10.4. Troubleshooting and Debugging a Stream Application 90
10.4.1. Monitoring SAM Apps and Identifying Performance Issues 90
10.4.2. Identifying Processor Performance Bottlenecks 95
10.4.3. Debugging an Application through Distributed Log Search 100
10.4.4. Debugging an Application through Sampling 102

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 analytics 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. The data producing app will send CSV events from each sensor to a single Kafka topic and will pass the schema name for the event as a Kafka event header. See Data Producer Application Generates Events.
3. Use NiFi to consume the events from the Kafka topic, and then route, transform, enrich, and deliver the data from the two streams to two separate Kafka topics.

   See NiFi: Create a Dataflow Application.
4. Connect to the two streams of data to perform analytics on the stream.

   See Creating and Configuring the Kafka Source 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.

   See Joining Multiple Streams.
6. Filter the stream on only events that are infractions or violations.

   See Filtering Events in a Stream using Rules.
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 perform analysis on the streaming data.

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

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.

   See Using Aggregate Functions over Windows.

9. When each of the preceding rules fires, create alerts, and make them instantly accessible.

   See Streaming Alerts to an Analytics Engine for Dashboarding.

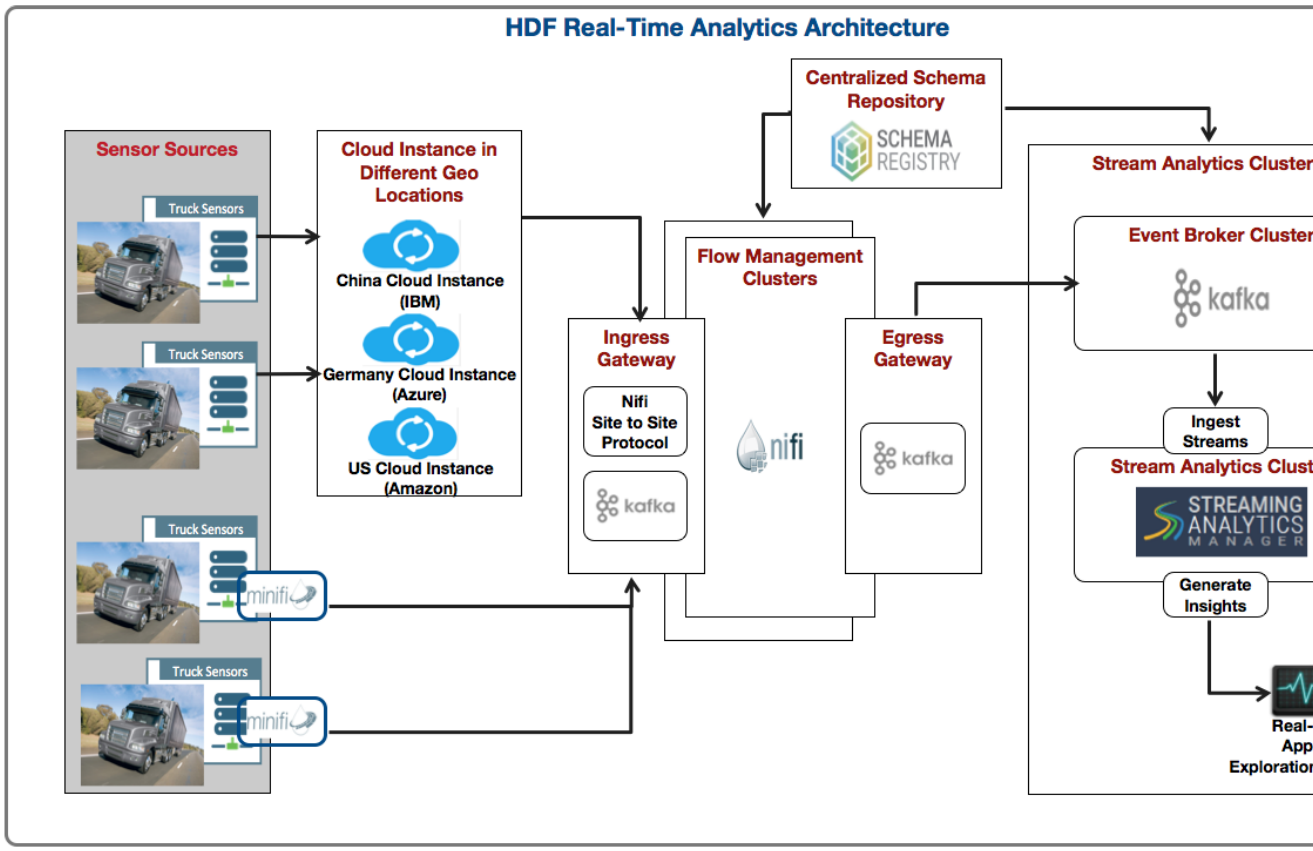
10. Execute a logistical regression Spark ML model on the events in the stream to predict if a driver is going to commit a violation. If violation is predicted, then generate an alert.

    See Advanced: Doing Predictive Analytics on the Stream.

The following sections walk you through how to implement all ten requirements. Requirements 1-3 are performed using NiFi and Schema Registry. Requirements 4 through 10 are implemented using the new Streaming Analytics 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.


HDF Real-Time Analytics Architecture
Centralized Schema Repository
SCHEMA REGISTRY
Sensor Sources
Truck Sensors
Truck Sensors
Truck Sensors
Truck Sensors
minifi
minifi
Cloud Instance in Different Geo Locations
China Cloud Instance (IBM)
Germany Cloud Instance (Azure)
US Cloud Instance (Amazon)
Ingress Gateway
Nifi Site to Site Protocol
kafka
Flow Management Clusters
nifi
Egress Gateway
kafka
Stream Analytics Cluster
Event Broker Cluster
kafka
Ingest Streams
Stream Analytics Clust
STREAMING ANALYTICS MANAGER
Generate Insights
Real-
App
Exploratior

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. Install a standalone HDF cluster.
3. Install a standalone HDP cluster.

You can find more information about your HDF and HDP cluster deployment options in *Planning Your Deployment*.

You can find more information about which versions of HDP and Ambari you should use with your version of HDF in the *HDF Support Matrix*.

More Information

Planning Your Deployment

HDF Support Matrix

2.2. Registering Schemas in Schema Registry

The trucking application streams CSV events from the two sensors to a single Kafka topic called `raw-all_truck_events_csv`. NiFi consumes the events from this topic, and then routes, enriches, and delivers them to a set of Kafka topics (`truck_events_avro` and `truck_speed_events_avro`) for consumption by the streaming analytics applications. To do this, you must perform the following tasks:

- Create the 3 Kafka topics
- Register a set of Schemas in 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 \
--partitions 3 \
--topic raw-all_truck_events_csv;

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

./kafka-topics.sh \
--create \
--zookeeper <zookeeper-host>:2181 \
--replication-factor 2 \
--partitions 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 two sensor streams and the two Kafka topics to which NiFi will publish the enriched events. Registering the Kafka topic schemas is beneficial 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.

   a. Enter the following:

   - `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`

b. Check the evolve check box.

c. Copy the schema from here and paste it into the Schema Text area.

d. 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.

a. Enter the following information:

- `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`

b. Check the evolve check box.

c. Copy the schema from here and paste it into the Schema Text area.

d. Click **Save**.

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

a. Enter the following information:

- `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`

b. Check the evolve checkbox.

c. Copy the schema from here and paste it into the Schema Text area.

d. 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:

a. Enter the following information:

- `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`

b. Check the evolve check box.

c. Copy the schema from here and paste it into the Schema Text area.

d. 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.

2.3. Setting up an Enrichment Store, Creating an HBase Table, and Creating an HDFS Directory

About This Task

To prepare for Performing Predictive Analytics on Streams, you need some HBase and Phoenix tables. This section gives you instructions on setting up the HBase and Phoenix tables timesheet and drivers, loading them with reference data, and downloading the custom UDFs and processors to perform 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. Name the unzipped folder $SAM_EXTENSIONS.

Steps for Creating Phoenix Tables and Loading Reference Data

1. Copy the $SAM_EXTENSIONS/scripts.tar.gz to a node where HBase/Phoenix client is installed.
2. On that node, untar the `scripts.tar.gz`. Name the directory $SCRIPTS.

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

3. Navigate to the directory where the phoenix script is located which will create the phoenix 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 add the script to JAVA_HOME.

```
./phoenix_create.sh
```

Steps for Verifying Data has Populated Phoenix Tables

1. Start up the sqlline Phoenix client.

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

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

2. List all the tables in Phoenix.

```
!tables
```

3. Query the drivers and timesheet tables.

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

Steps for Starting HBase and Creating an HBase Table

1. This can be easily done by adding the HDP HBase Service using Ambari.

2. Create a new HBase table by logging into an node where Hbase client is installed then execute the following commands:

```

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

/hbase shell

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

Steps for Creating an HDF Directory

Create the following directory in HDFS and give it access to all users.

1. Log into a node where HDFS client is installed.

2. Execute the following commands:

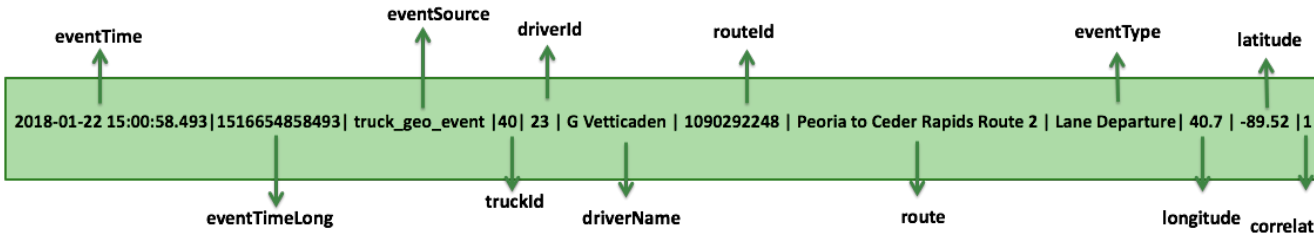
```
su hdfs

hadoop fs -mkdir /apps/trucking-app

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

3. Creating a Dataflow Application

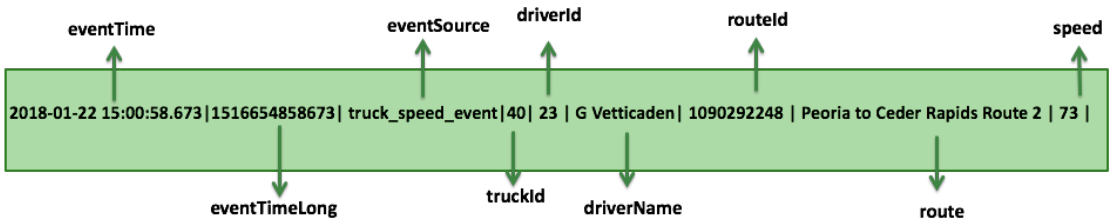
3.1. Data Producer Application Generates Events

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

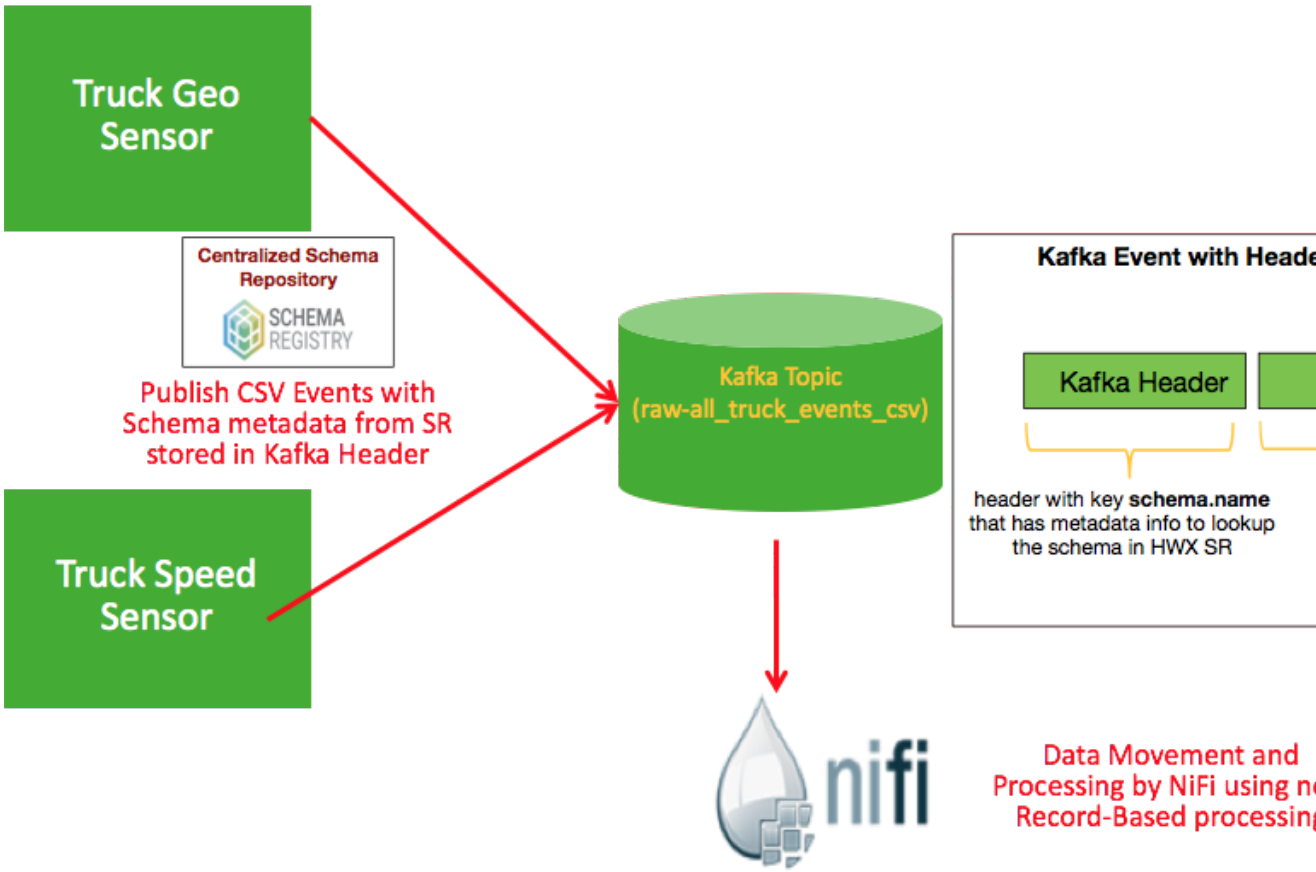
Geo Sensor Stream



Speed Sensor Stream



The date producing application or data simulator publishes CSV events with schema name in the Kafka event header (leveraging the Kafka Header feature in Kafka 1.0). The following diagram illustrates this architecture:



Use NiFi's Kafka 1.0 ConsumeKafkaRecord to do the following:

1. Grab the schema name from Kafka Header and store in flow attribute called schema.name
2. Use the schema name to look up the Schema in HWX SR
3. Use the schema from HWX SR to convert to ProcessRecords
4. Route events by event source (speed or geo) using SQL

The below sections walks through how to setup this data flow.

3.2. NiFi: Create a Dataflow Application

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

   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`.

2. Edit the properties to configure the Schema Registry URL based on your environment.

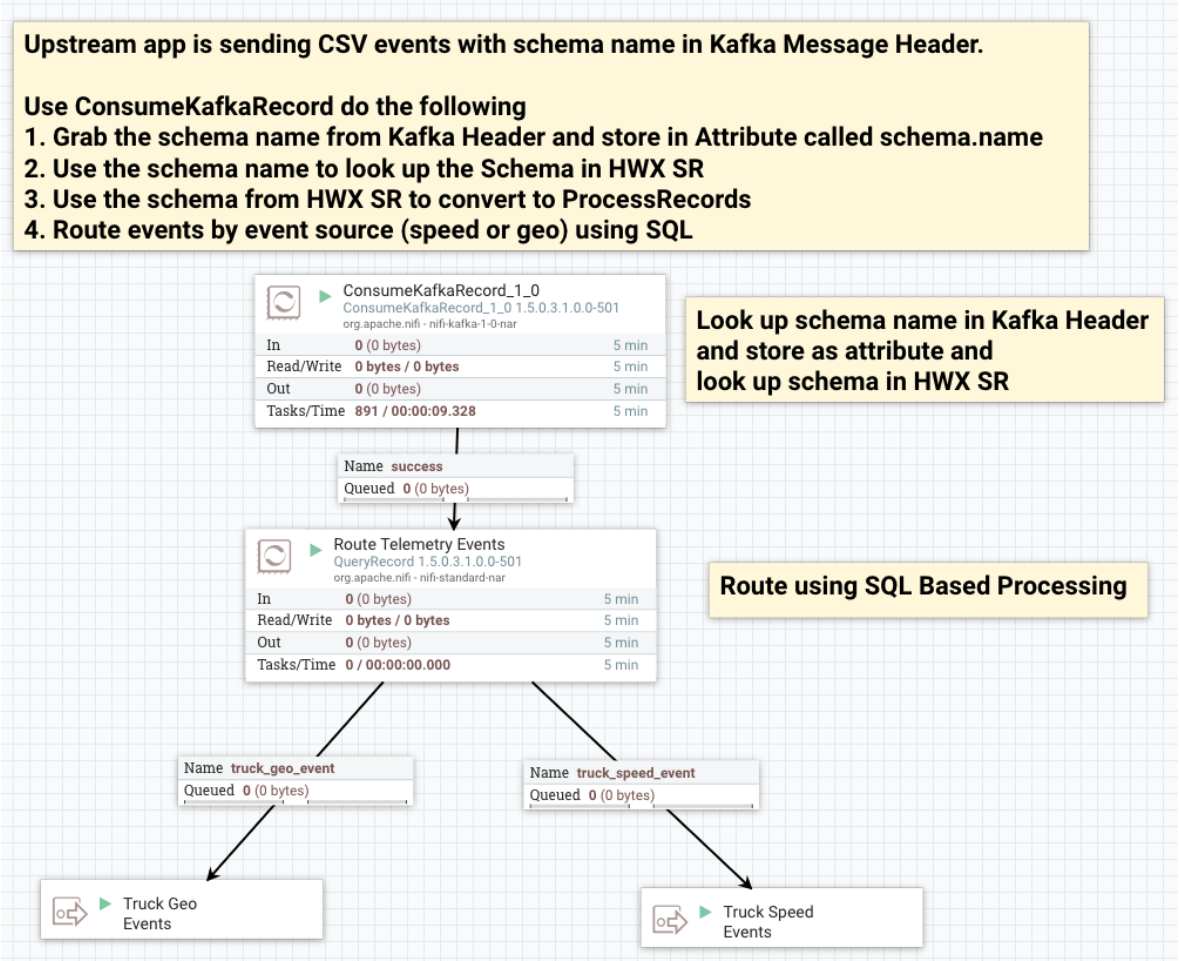
   You can find this value in the Streaming Analytics Manager Service in Ambari for the configuration property called `registry.url`. The URL should look similar to the following: `http://$REGISTRY_SERVER:7788/api/v1`.

3. Enable the HWX Schema Registry and the other controller services. For more information on the RecordReader and RecordWriter controller services, see the Schema Registry User Guide. You should have 5 controller services enabled like the following.

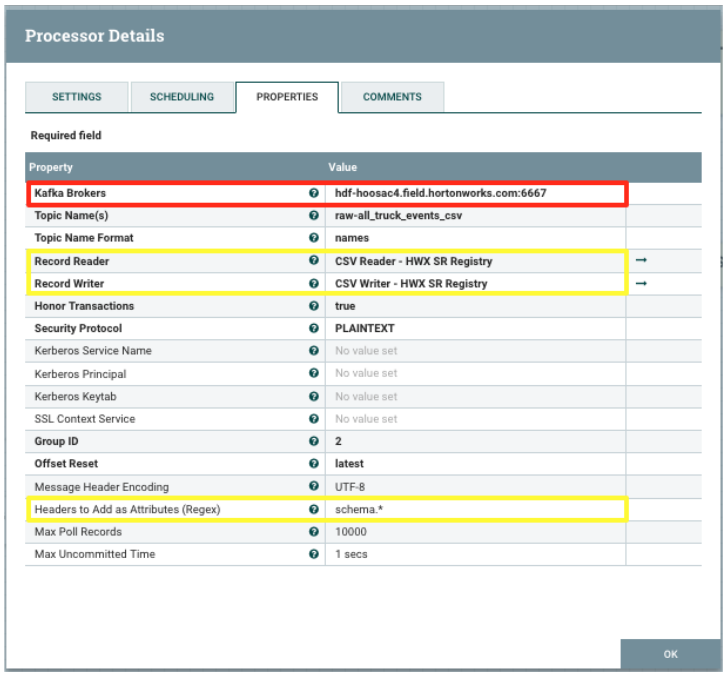
	Name	Type	Bun...	State	Scope
	HWX Schema Registry	HortonworksSchema...	org...	Enabled	NiFi Flow
	Enrich- ReverseGeoCodeLookupService	ScriptedLookupServic...	org...	Enabled	NiFi Flow
	CSV Writer - HWX SR Registry	CSVRecordSetWriter 1...	org...	Enabled	NiFi Flow
	CSV Reader - HWX SR Registry	CSVReader 1.5.0.3.1.0...	org...	Enabled	NiFi Flow
	Avro Writer - HWX SR Registry - HWX Content Enc...	AvroRecordSetWriter ...	org...	Enabled	NiFi Flow

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 csv events from the Kafka topic called raw-all_truck_events_csv. We will use the new Kafka 1.0 ConsumerKafkaRecord processor for this.



Configure the 'Kafka Brokers' value in ConsumeKafkaRecord_1_0 based on your cluster.



Kafka 1.0 ConsumerKafkaRecord will do the following:

1. Grab the schema name from Kafka Header and store in Attribute called schema.name (see property called 'Headers to Add as Attributes').

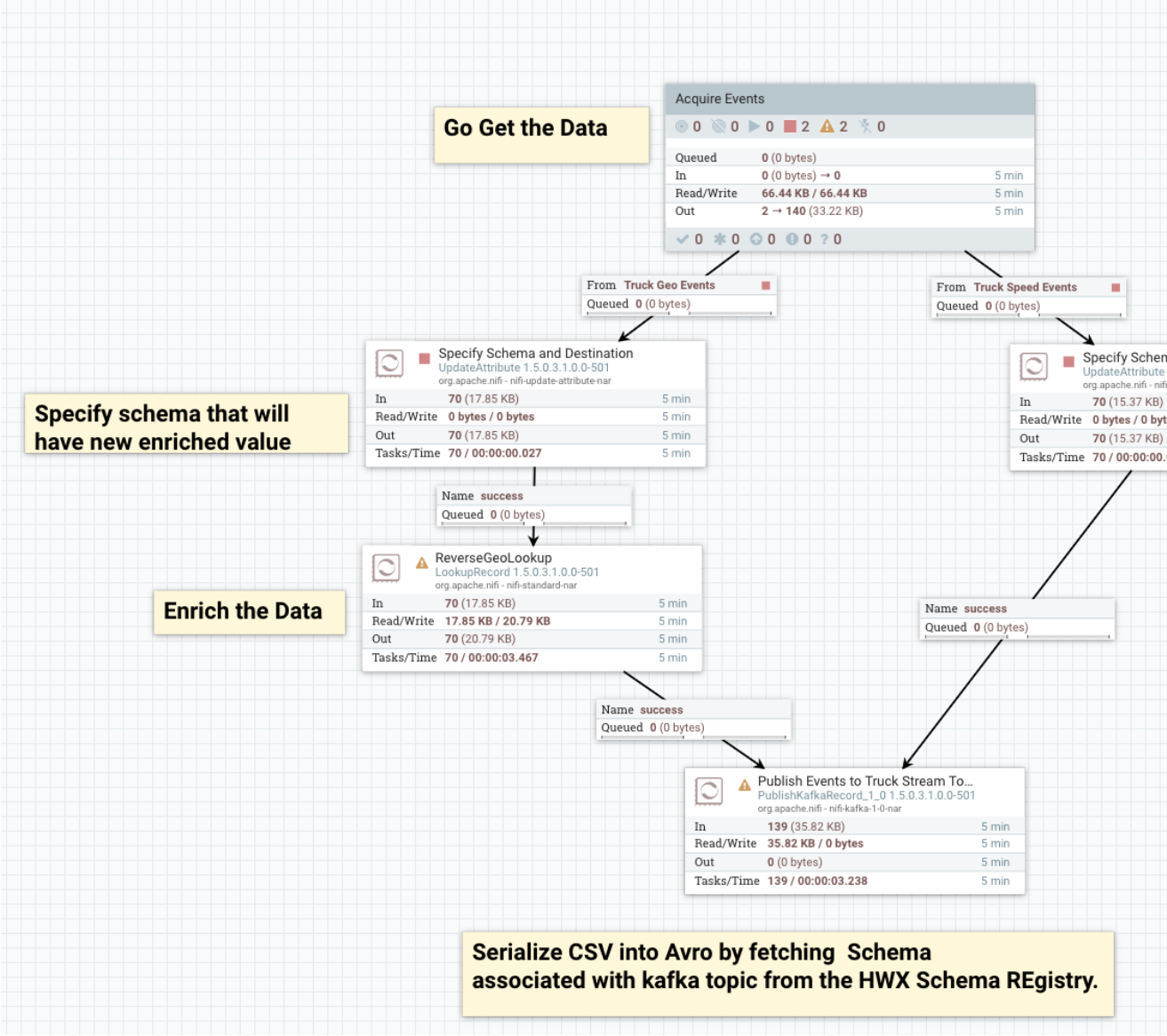
2. Use the schema name to look up the Schema in HWX Schema Registry.

3. Use the schema from HWX SR to convert to ProcessRecords.

4. Route events by event source (speed or geo) using SQL.

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

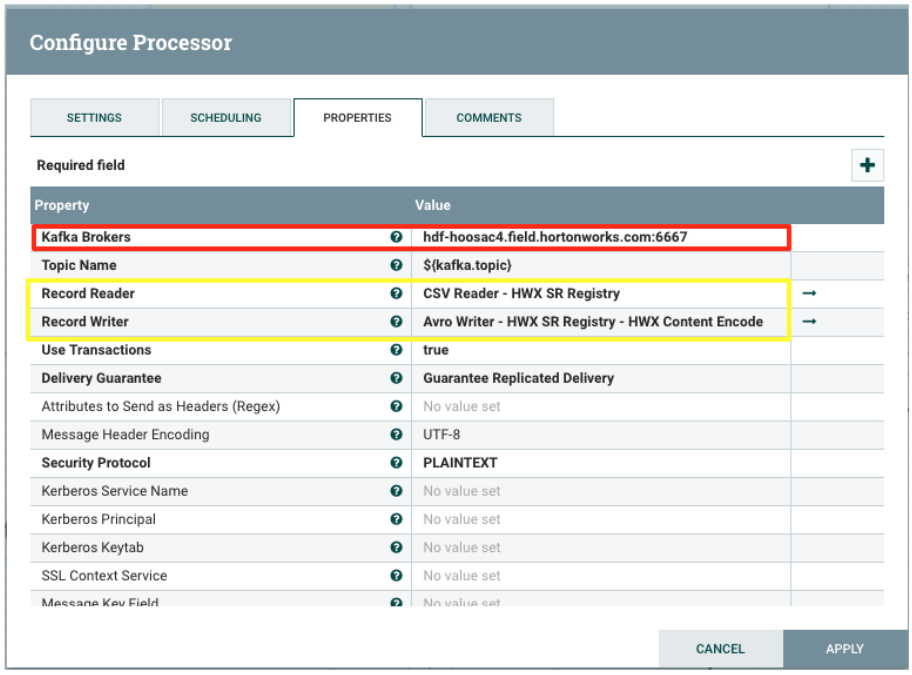
After NiFi has done the routing, transforms, and enrichment, NiFi will publish the enriched events into Kafka topics. These topics have a schema registered for it in the Schema Registry and we will store the schema identifier for the schema in the FlowFile attributes (UpdateAttribute processors) and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



The PublishKafkaRecord_1_0 processor to push the events into Kafka serialized as Avro objects with HWX Schema encoded into the payload so that SAM can process it. Make sure for the PublishKafkaRecord, you change the Kafka Brokers property value to your cluster settings.

Note

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



3.2.4. Start the NiFi Flow

Start the Process Grouped called "Use Case 1".

4. Creating a Stream Analytics Application

1. Creating a Service Pool and Environment [18]
2. Creating Your First Application [18]
3. Creating and Configuring the Kafka Source Stream [19]
4. Connecting Components [21]
5. Joining Multiple Streams [21]
6. Filtering Events in a Stream using Rules [23]
7. Using Aggregate Functions over Windows [24]
8. Implementing Business Rules on the Stream [25]
9. Transforming Data using a Projection Processor [27]
10. Streaming Alerts to an Analytics Engine for Dashboarding [29]
11. Streaming Violation Events to an Analytics Engine for Descriptive Analytics [30]
12. Streaming Violation Events into a Data Lake and Operational Data Store [32]

4.1. Two Options for Creating the Streaming Analytics Applications

Over the next few sections, we walk through building up the stream analytics app to implement all the requirements of this use case. This will entail actions such as:

- Uploading Custom UDFs
- Uploading Custom Sources
- Uploading Custom Sinks
- Uploading a PMML model into Model Registry
- Uploading Custom Processors
- Creating Service Pools for HDP and HDF
- Create SAM Environment required for the Reference App
- Building the Reference application with streaming joins, filtering, aggregations over windows, dashboarding, execute PMML models, doing streaming split pattern, etc..
- Setting up Test Cases for the Reference App

There are two options to performing these actions:

1. Doing all of these steps manually as the subsequent sections will walk you through. This is recommended if you are new to SAM and want to build a complex app from scratch.

2. Running a utility that performs all of these actions for you.

4.1.1. Setting up the Stream Analytics App using the TruckingRefAppEnvEnviornmentBuilder

About This Task

Follow the below instructions if you want to set up the reference application using an utility. If not, skip this section and go through the next set of sections. Perform the below steps on the host where SAM is running.

Steps

1. Download the SAM_EXTENSIONS zip file. Unzip the contents. Call the unzipped folder `$SAM_EXTENSION`

2. Navigate to the unzip location:

```
cd $SAM_EXTENSION/ref-app
```

3. Modify the `trucking-advanced-ref-app.properties` file based on your environment:

   - sam.rest.url = the REST url of the SAM instance in your env (e.g.: http://<SAM_HOST>:<SAM_PORT>/api/v1)
   - sam.service.pool.hdf.ambari.url = The rest endpoint for the HDF cluster you installed (e.g.: http:/<HDF_AMBARI_HOST>:<PORT>/api/v1/clusters/<cluster_name>)
   - sam.service.pool.hdp.ambari.url = The rest endpoint for the HDP cluster you installed. (e.g.: http:/<HDP_AMBARI_HOST>:<PORT>/api/v1/clusters/<cluster_name>)
   - sam.schema.registry.url = The url of the Schema Registry service in SAM you installed as part of the HDF cluster (e.g.: http://SR_HOST:SR_PORT/api/v1)

4. Run the following command

   - java -cp sam-trucking-ref-app-shaded.jar hortonworks.hdf.sam.refapp.trucking.env.TruckingRefAppEnviornmentBuilderImpl trucking-advanced-ref-app.properties

If script ran successfully, you should see output like the following (it will take about 3-5 minutes to finish):

```
Trucking Ref App environment creation time[367 seconds] Trucking Ref App SAM
 URL: http://SAM_HOST:SAM_PORT/#/applications/78/view
```

4.1.2. Configuring and Deploying the Reference Application

In SAM, go into the edit mode the Trucking Ref App using the url outputted in the logs and then follow the below steps to configure and deploy the ref app:

1. Double click on the TruckGeoEvent Kafka Source and configure it.
   a. Select a cluster.
   b. Select truck_events_avro for the kafka topic.
   c. Select 1 for the Reader Schema Version
   d. Put an unique value for the consumer group id
2. Double click on the TruckSpeedEvent Kafka Source and configure it.
   a. Select a cluster.
   b. Select truck_speed_events_avro for the kafka topic.
   c. Select 1 for the Reader Schema Version
   d. Put an unique value for the consumer group id
3. Open the configuration for the two Druid sinks (Alerts-Speeding-Driver-Cube and Dashboard-Predictions) and configure each one by selecting a cluster.
4. Open the configuration for the ENRICH-HR and ENRICH-Timesheet configure the 'Phoenix ZooKeeper Connection Url" based on your cluster (e.g.: ZK_HOST:ZK:PORT).
5. Open the configuration for the Hbase and HDFS sink and ensure that the cluster is selected.

4.2. Creating 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.

4.3. Creating Your First Application

About This Task

The Streaming Analytics Manager provides capabilities to the application developer for building streaming applications. You can go to the Stream Builder UI by selecting 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.

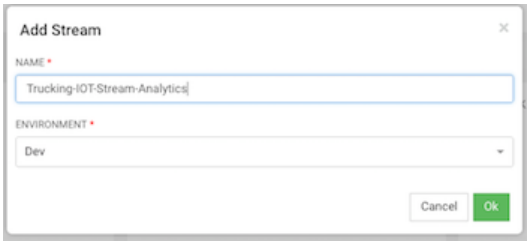
Steps

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 to stream.

Note

The name of the stream application cannot have any spaces.



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.

4.4. Creating and Configuring the Kafka Source Stream

About This Task

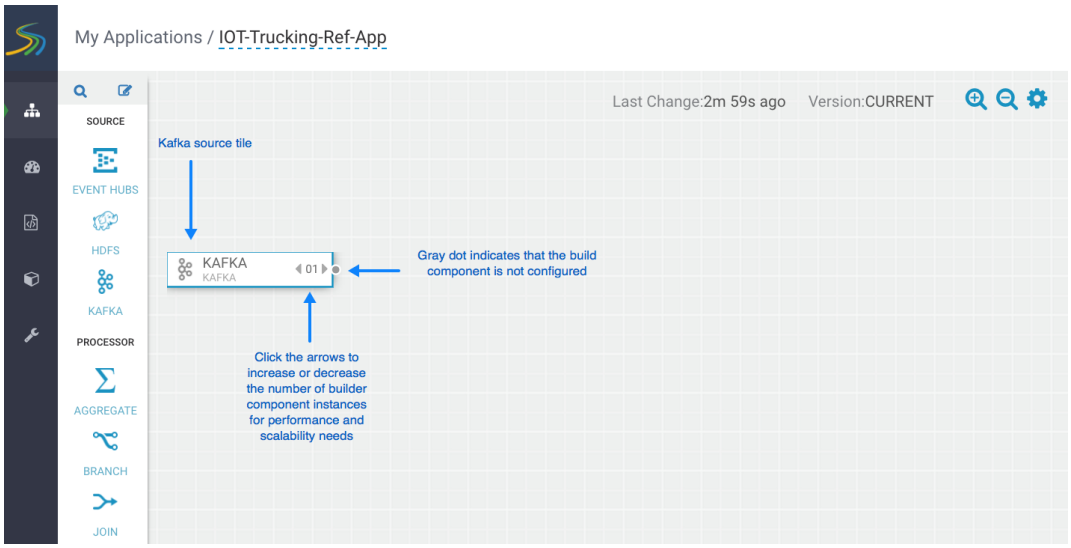
The first step in building a stream application is to drag builder components to the canvas. As described in the *Hortonworks DataFlow Overview*, Stream Builder offers four types of builder components: sources, processors, sinks, and custom components.

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.

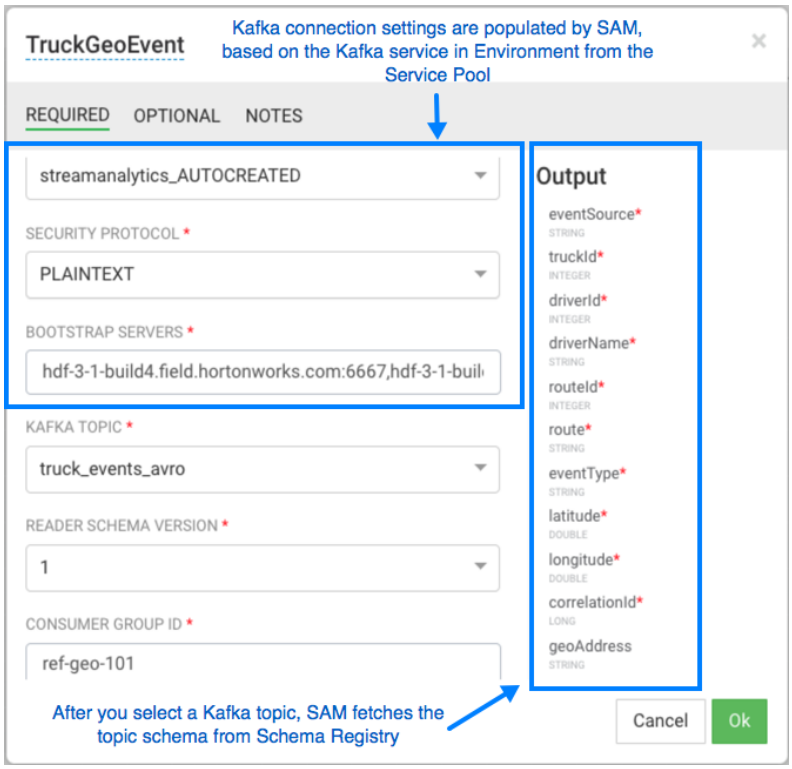
Steps

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



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:



Result

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

More Information

Hortonworks DataFlow Overview

4.5. 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.

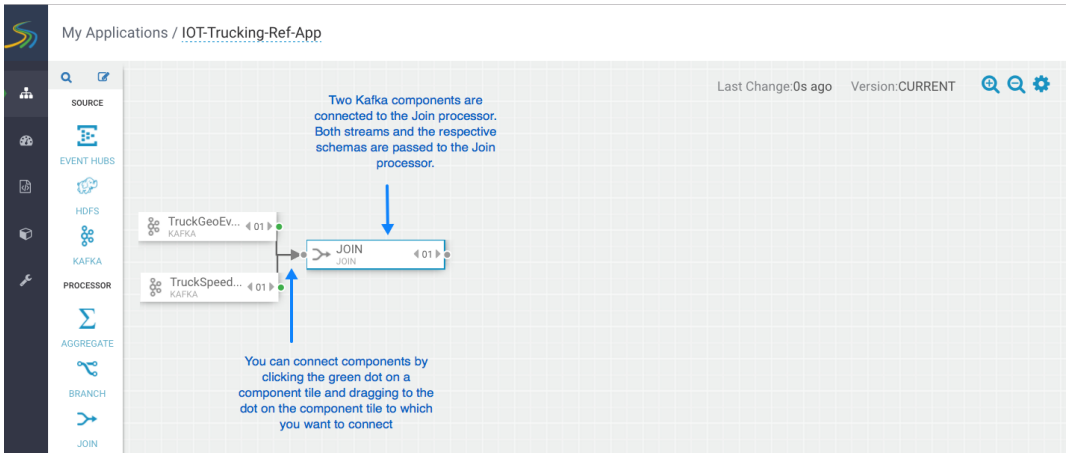
Steps

1. Click the green dot to the right 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.

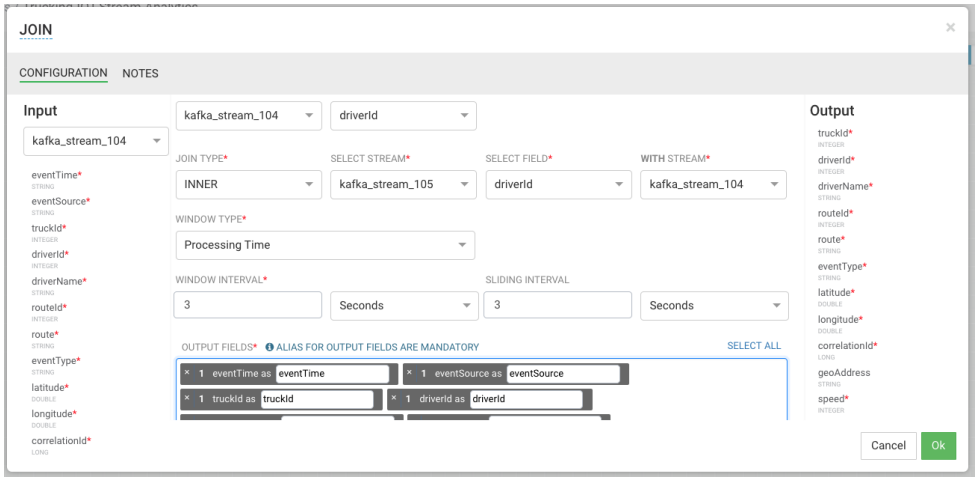


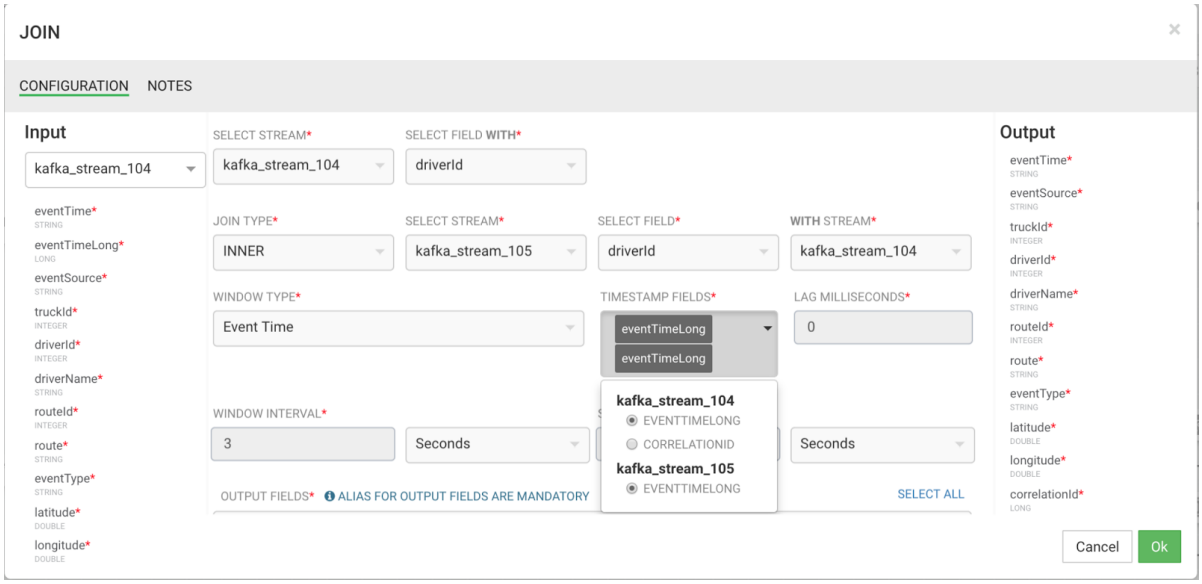
4.6. Joining Multiple Streams

About This Task

Joining multiple streams is an important SAM capability. You accomplish this by adding the Join processor to your stream application. You can join on multiple streams using one of two concepts available in the **Window Type** filed:

- Processing Time – Represents window based on when the event is processed by SAM
- Event Time – Represents the window based on the timestamp of the event.





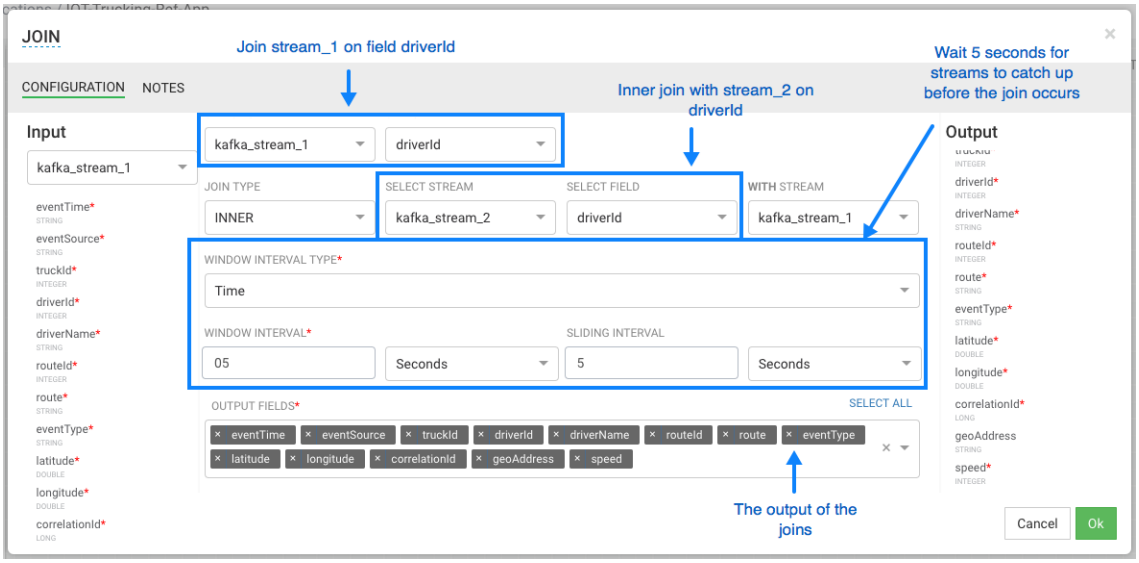
For the purposes of this reference application, use processing time.

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



4.7. 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.

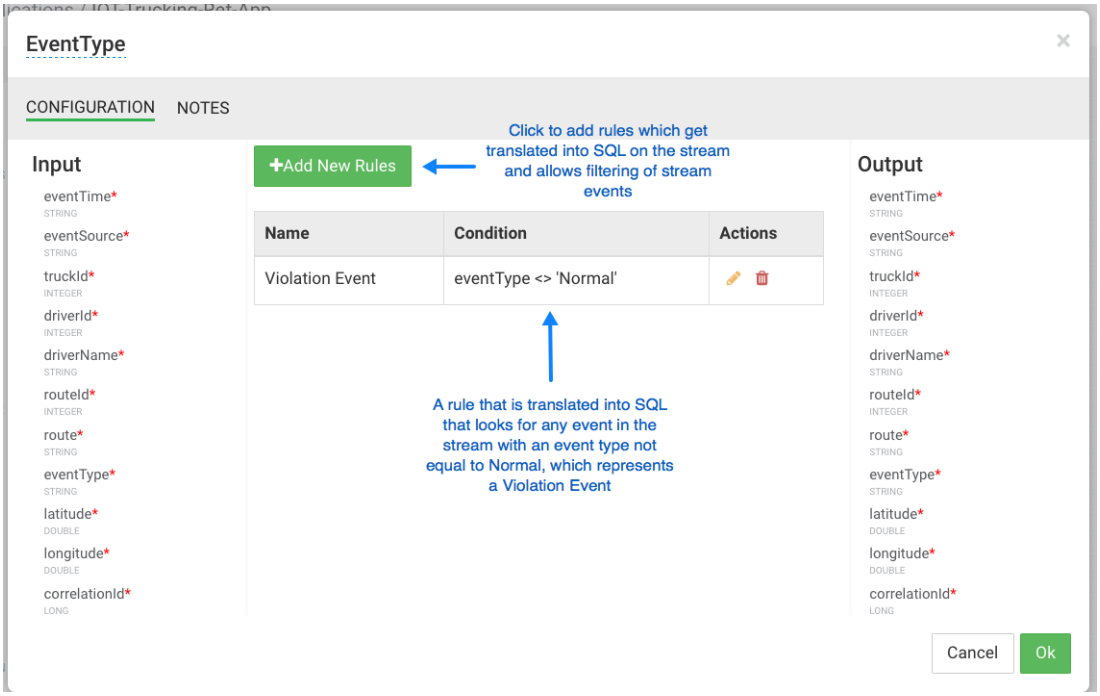
The following steps demonstrate this, implementing Requirement 6 of the use case.

Steps

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

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



4.8. Using Aggregate Functions over Windows

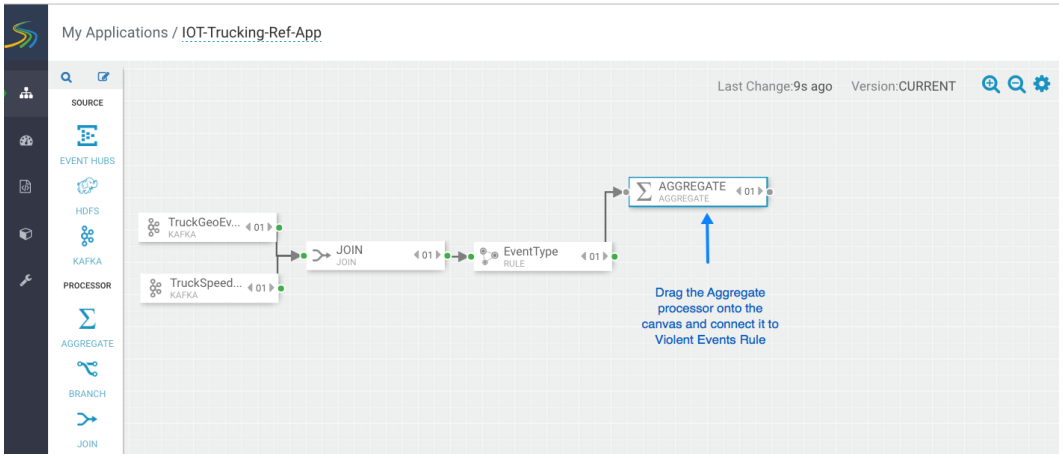
About This Task

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.

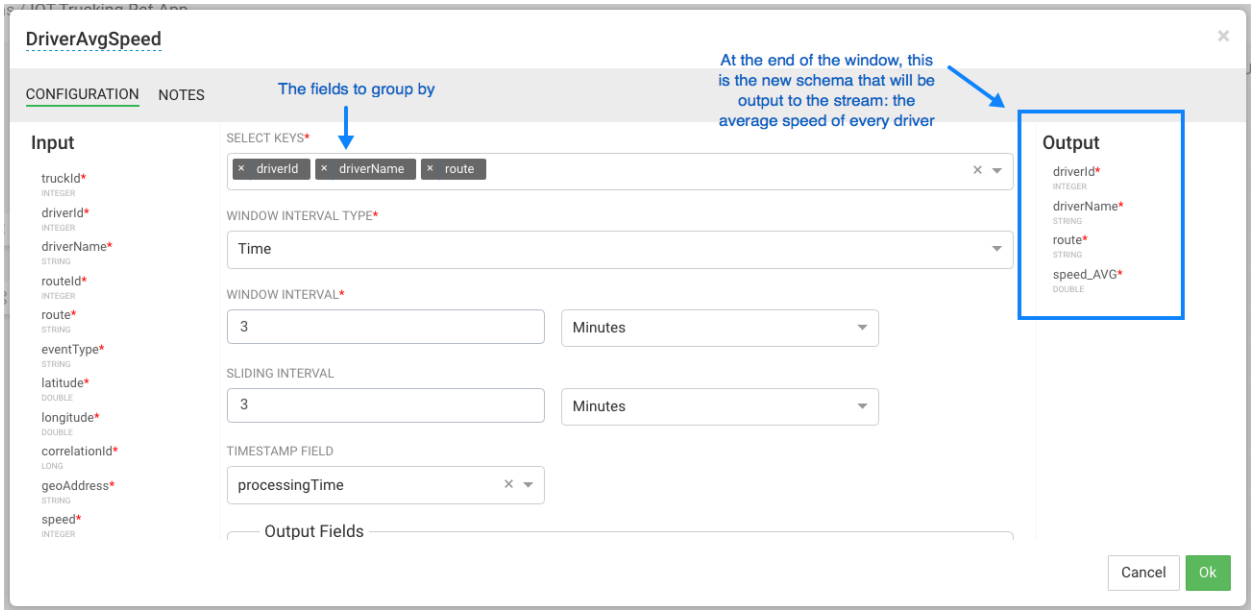
The following images show how to use the Aggregate processor to implement Requirement 8 of the use case.

Steps

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



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



4.9. Implementing Business Rules on the Stream

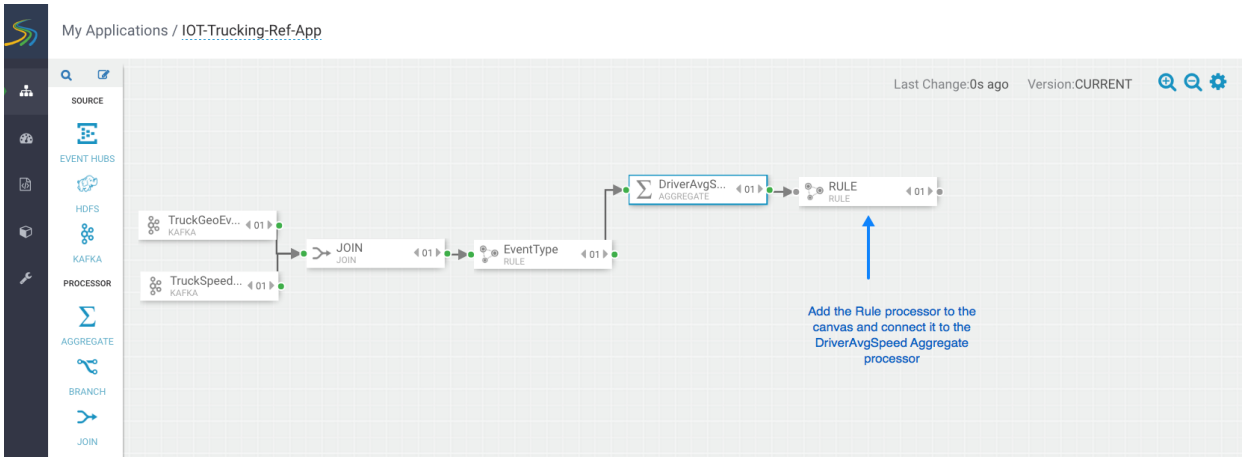
About This Task

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.

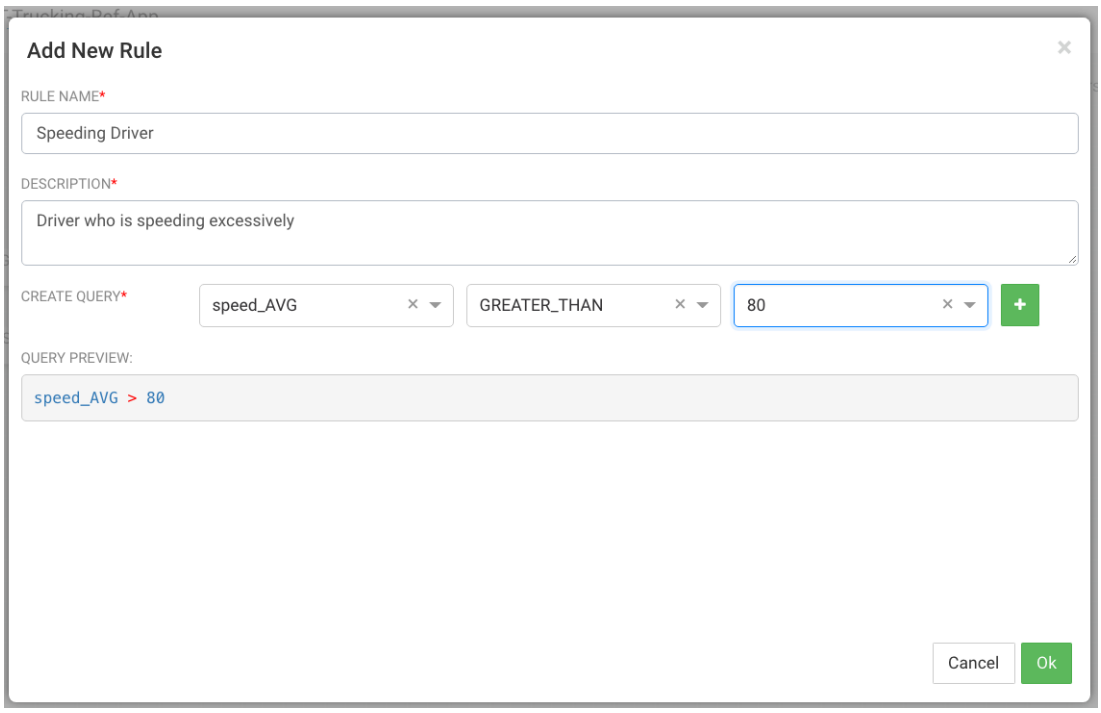
This step partially implements Requirement 8 of the use case.

Steps

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

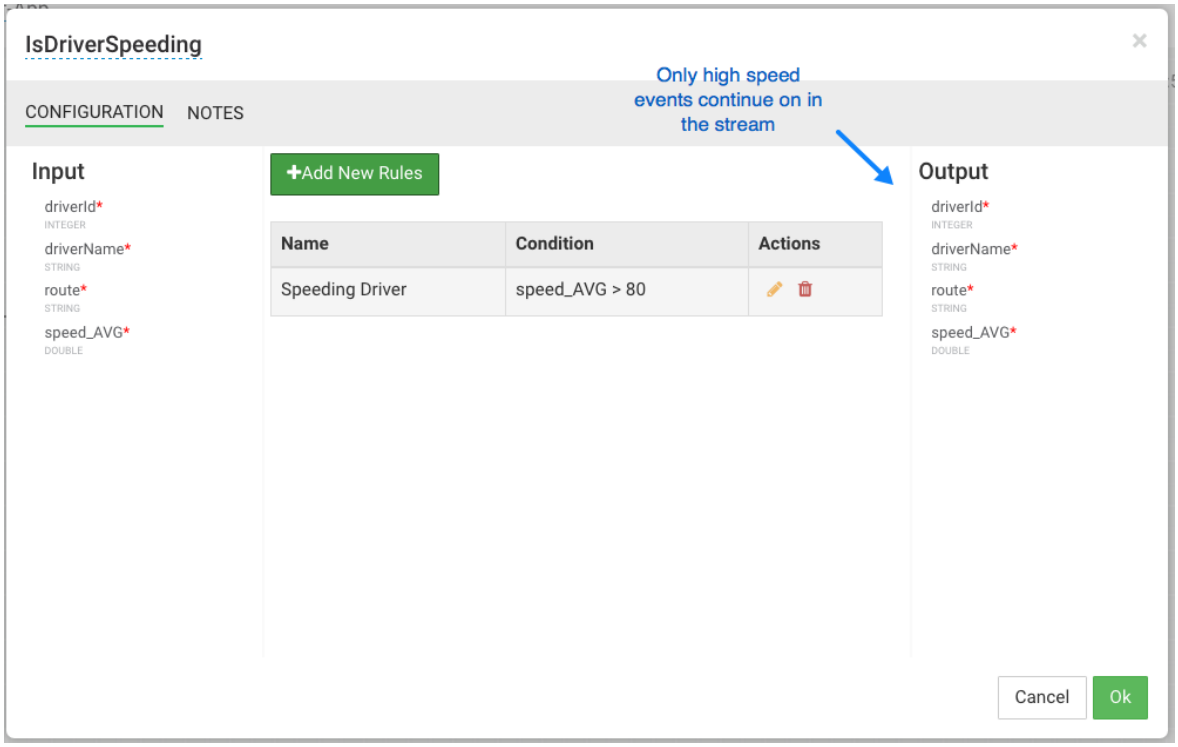


2. Configure the business rule as follows:



Result

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



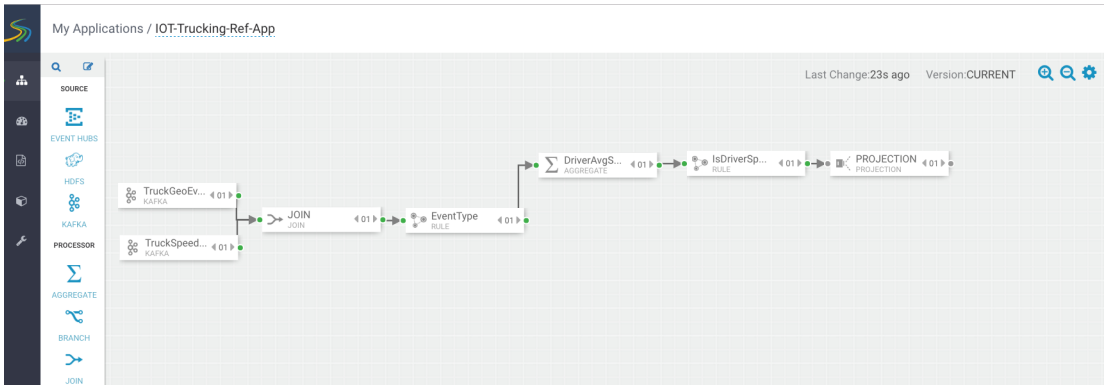
4.10. Transforming Data using a Projection Processor

About This Task

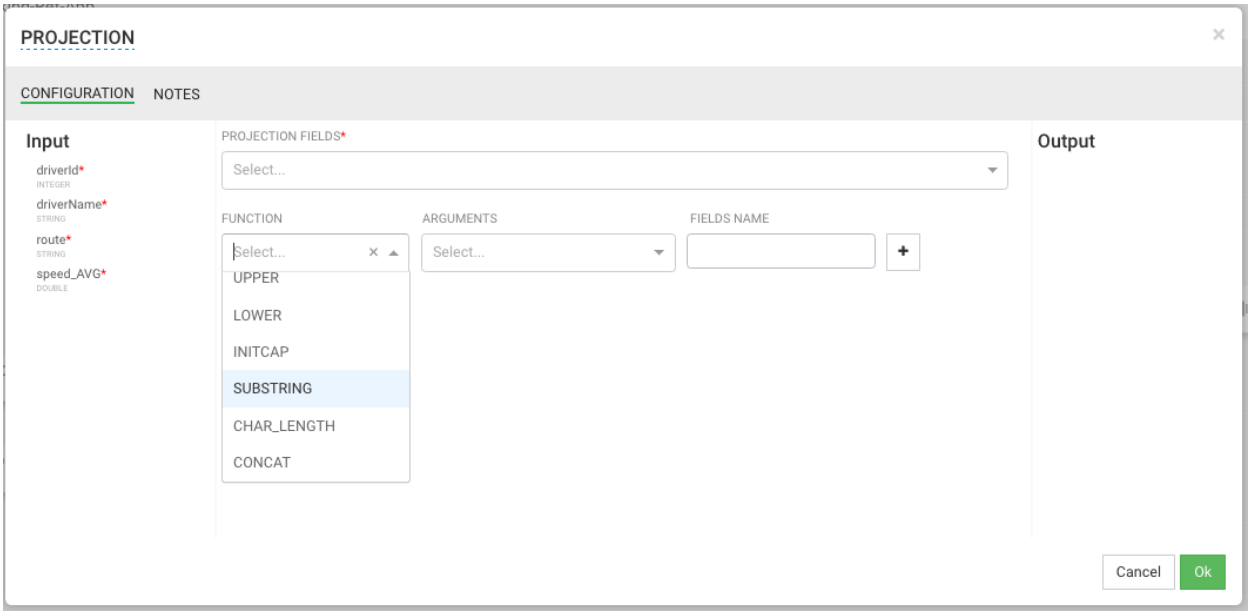
It is common to do transformations on the events in the stream. In our case, before we alert on the speeding driver, we want to convert the average speed we calculated in the aggregate processor into a integer from a double so it is easier to display in the alert. The projection processor allows you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

Steps

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



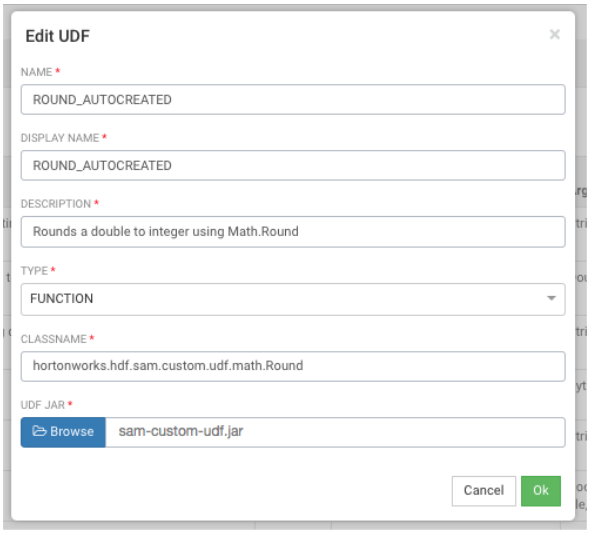
2. When you double-click on the projection processor, you see a number of out-of-the-box functions, however a Round function does not exist.



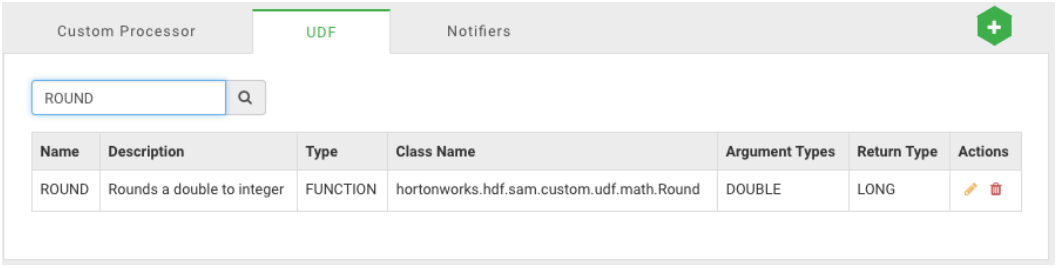
3. Adding UDFs (User Defined Functions) is easy to do within SAM. Follow the below steps to add Round UDF function to SAM.

   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, located in the `custom-udf` folder. 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:



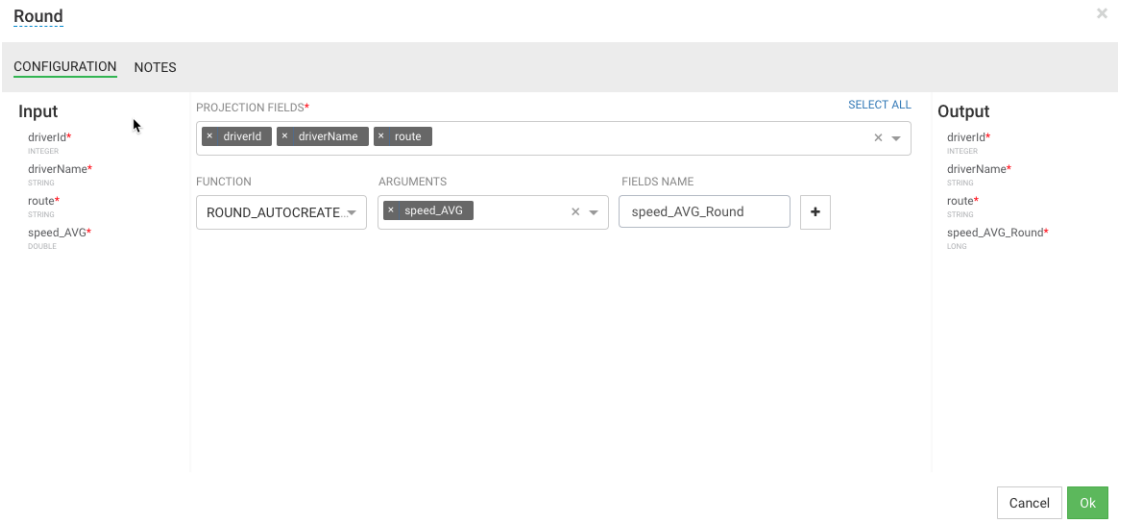
   c. After uploading the UDF, you should see the new Round UDF created.



4. After creating the UDF, go back to your Application and double-click Projection Processor you added to the canvas.

   You will see ROUND_AUTOCREATE in the **FUNCTION** drop down list.

5. Configure the ROUND_AUTOCREATE function as the following:



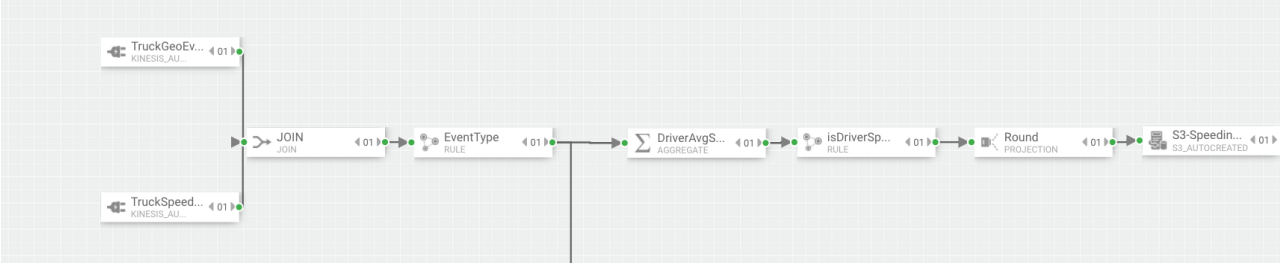
4.11. Streaming Alerts to an Analytics Engine for Dashboarding

About This Task

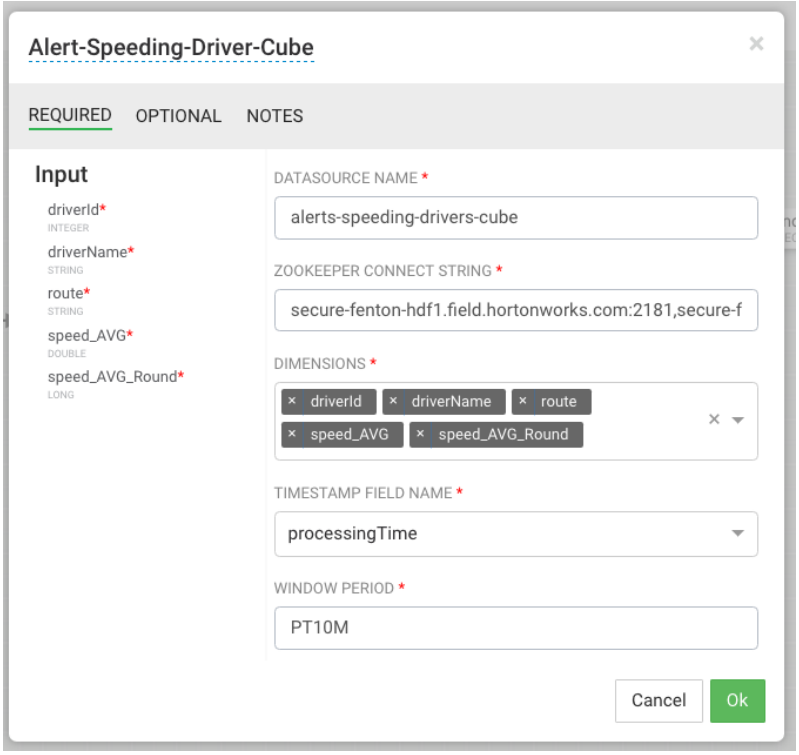
In addition to creating notification alerts, a common use case requirement is to send these alerts to a dashboard so they can be displayed and visualized. SAM offers this capability by allowing you to stream data into DRUID and then using Superset to create dashboards and visualizations.

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:



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

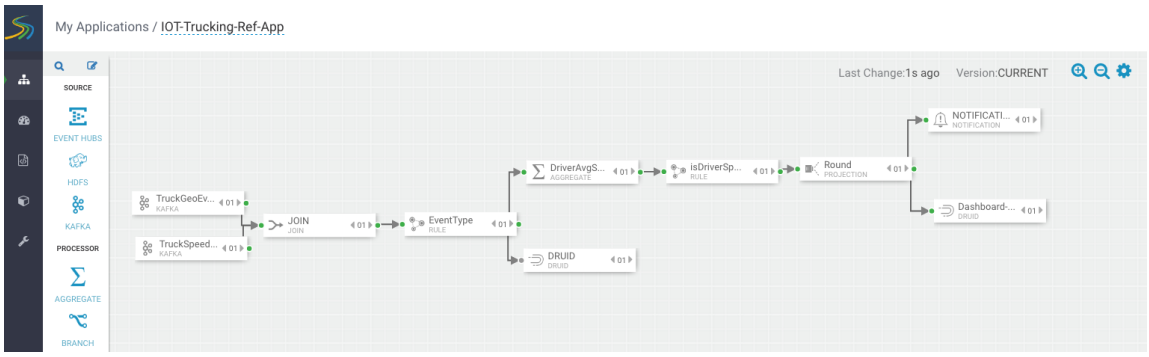
Now let's implement Requirement 7:

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.

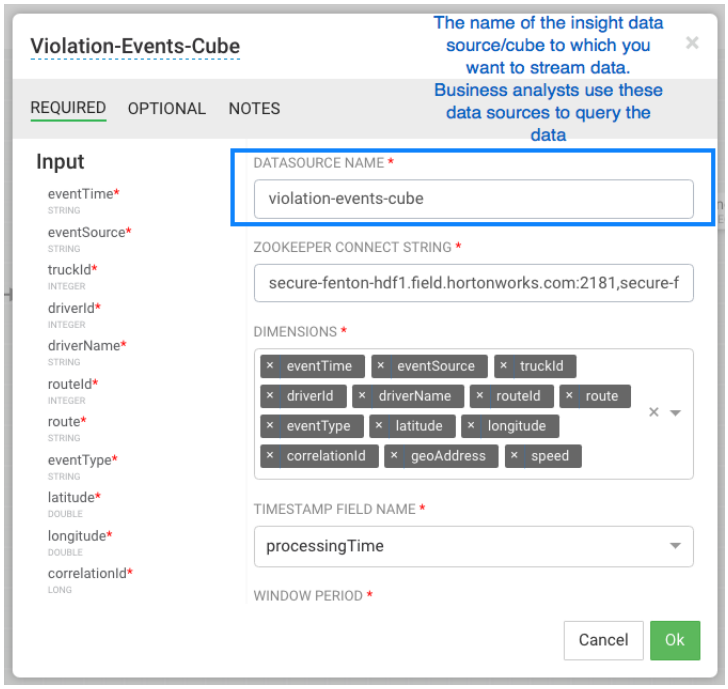
Steps

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

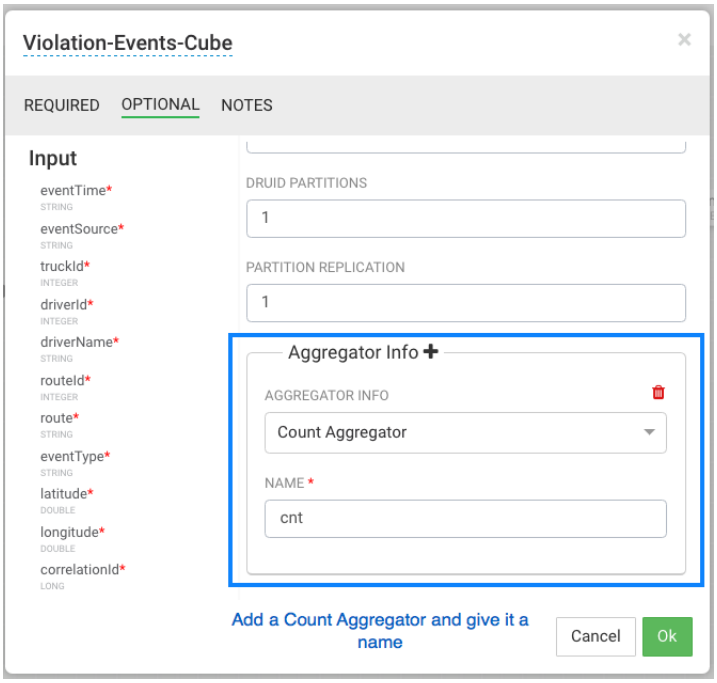


2. Configure the Druid processor.

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



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



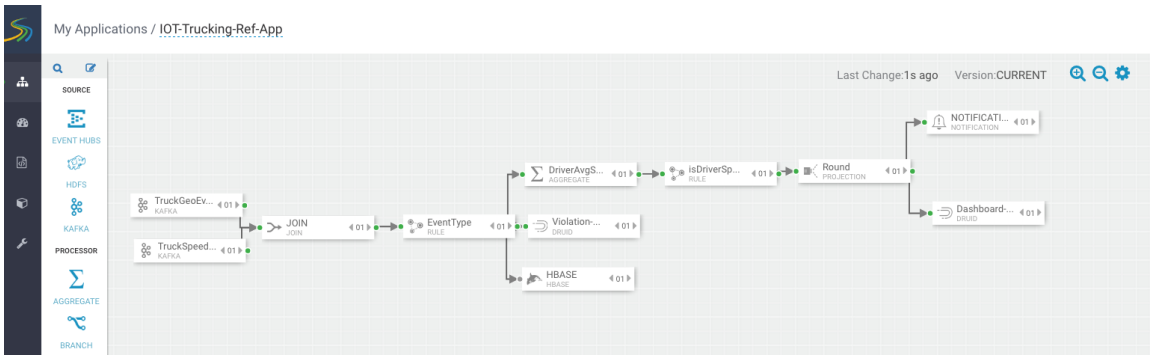
4.13. Streaming Violation Events into a Data Lake and Operational Data Store

About This Task

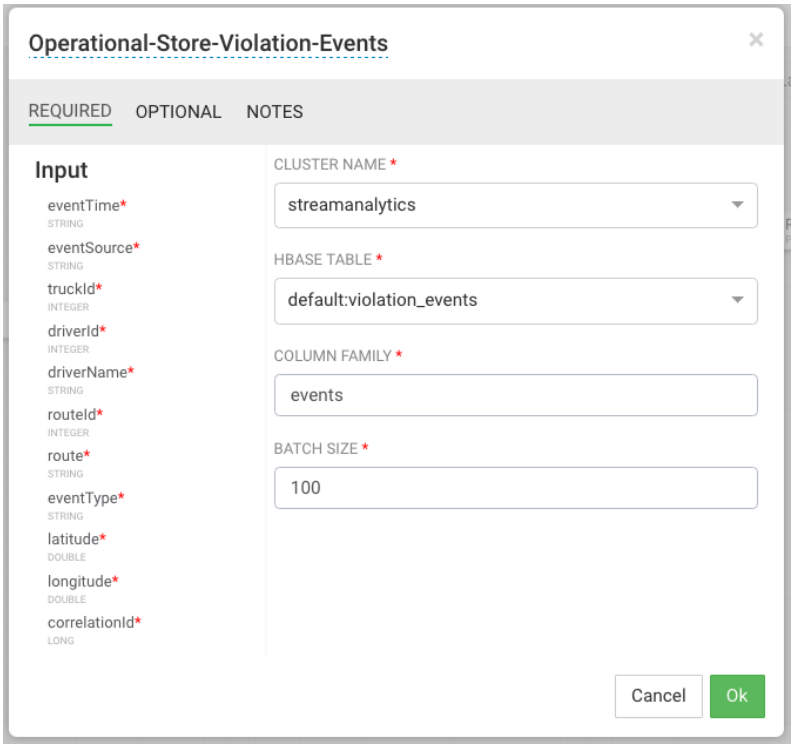
Another common requirement is to stream data into an operational data store like HBase to power real-time web applications as well as a data lake powered by HDFS for long term storage and batch ETL and analytics.

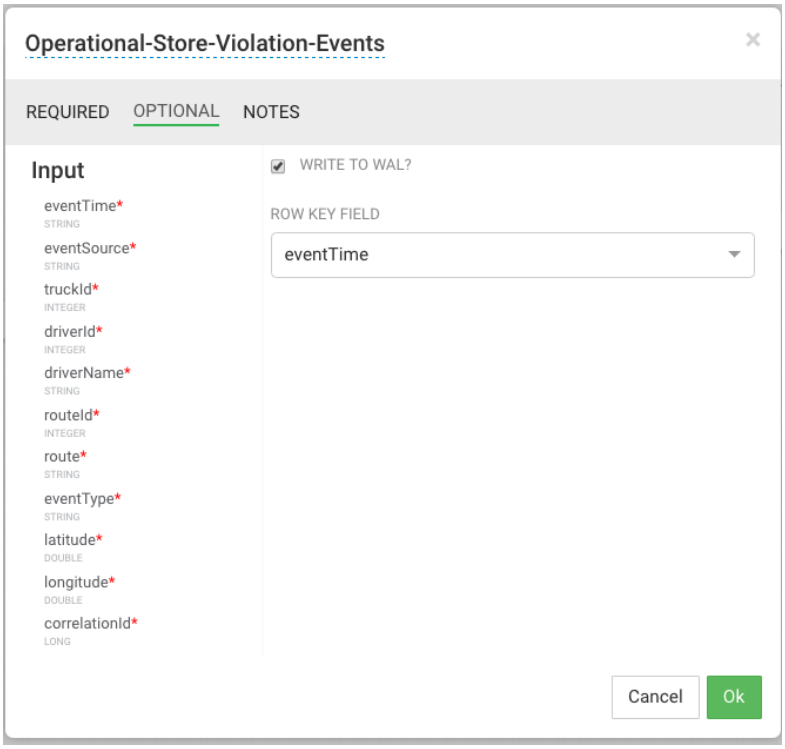
Steps

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

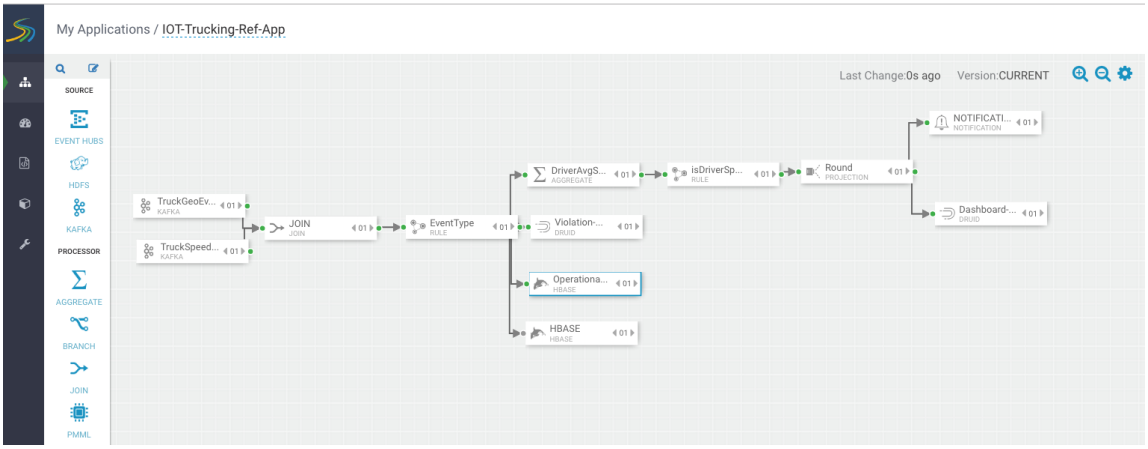


2. Configure the Hbase Sink using the following parameters.



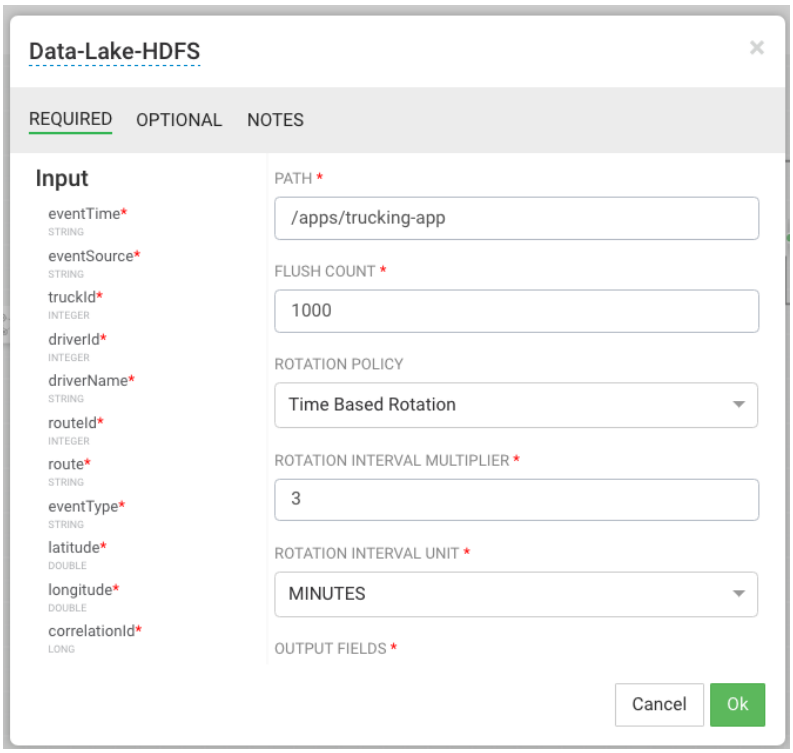


3. Drag the HDFS sink to the canvas and connect it to the ViolationEvents Rule processor.



4. Configure HDFS as below.

   Make sure you have permission to write into the directory you have configured for HDFS path.



5. Deploy an Application

5.1. Configure Deployment Settings

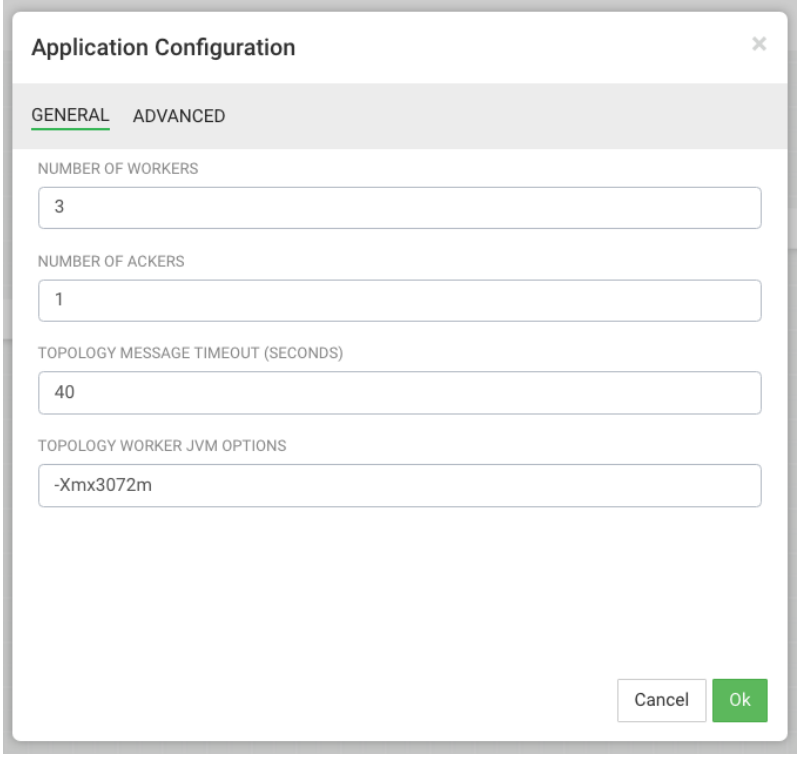
About This Task

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

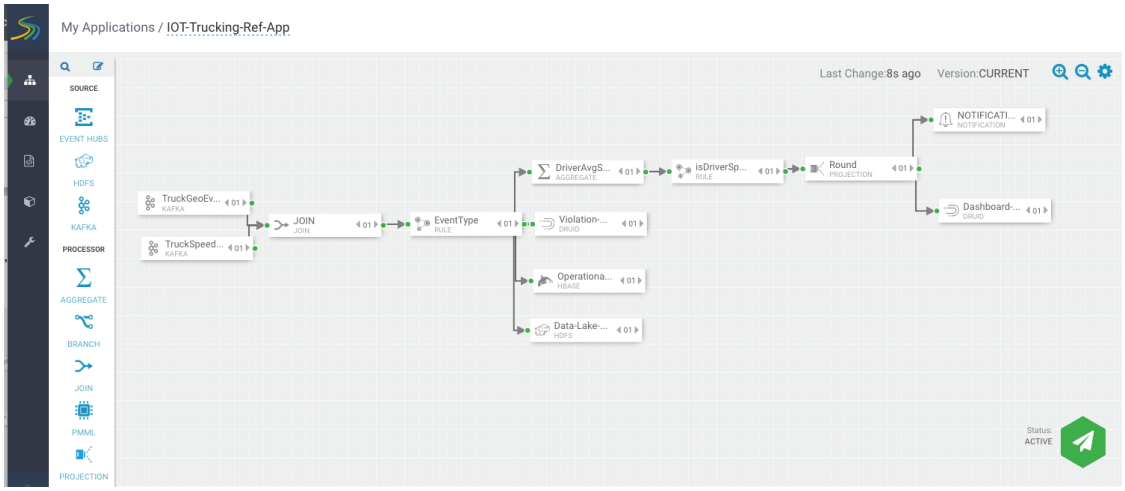
1. Click the gear icon at the top right corner of the canvas to display the **Application Configuration** dialog.
2. Increase **Number of Workers** to 5.
3. Set **Topology Worker JVM Options** to -Xmx3072m.

Example



5.2. Deploy the App

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



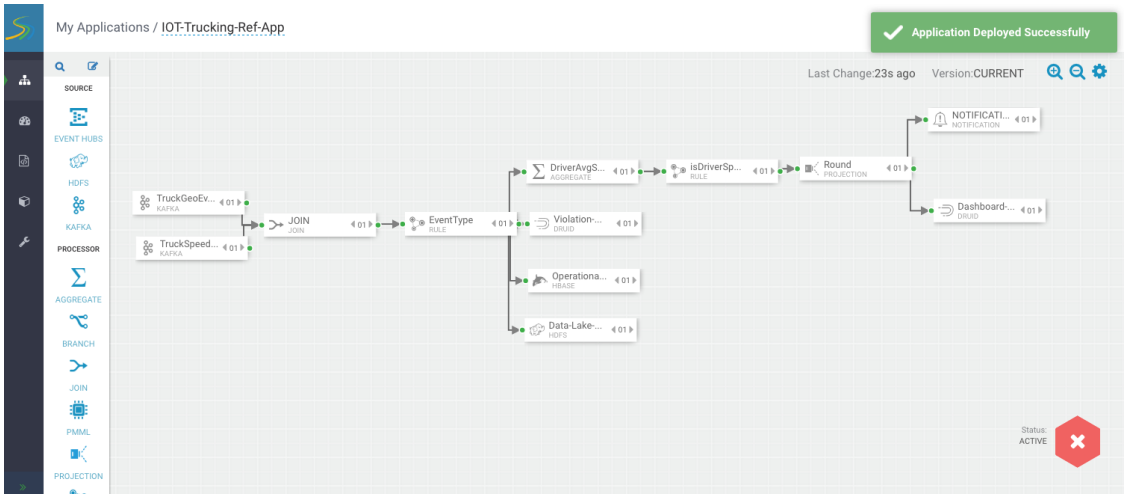
During the deployment process, Streaming Analytics Manager completes the following tasks:

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.

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, you can 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.
2. Unzip the Data Loader file and copy it to the cluster. Name the directory to which you unzipped the file $DATA_LOADER_HOME.
3. Execute the following commands.

   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.hdf.sam.refapp.trucking.simulator.SimulationRunnerApp \
20000 \
hortonworks.hdf.sam.refapp.trucking.simulator.impl.domain.transport.Truck \
hortonworks.hdf.sam.refapp.trucking.simulator.impl.collectors.
KafkaCSVEventWithSchemaHeaderCollector \
1 \
$DATA_LOADER_HOME/routes/midwest/ \
10000 \
$KAFKA_BROKER_HOST:$KAFKA_PORT \
ALL_STREAMS \
NONSECURE &
```

   You should see events being published into the Kafka topics called: raw-truck_events_avro and raw-truck_speed_events_avro. NiFi should consume

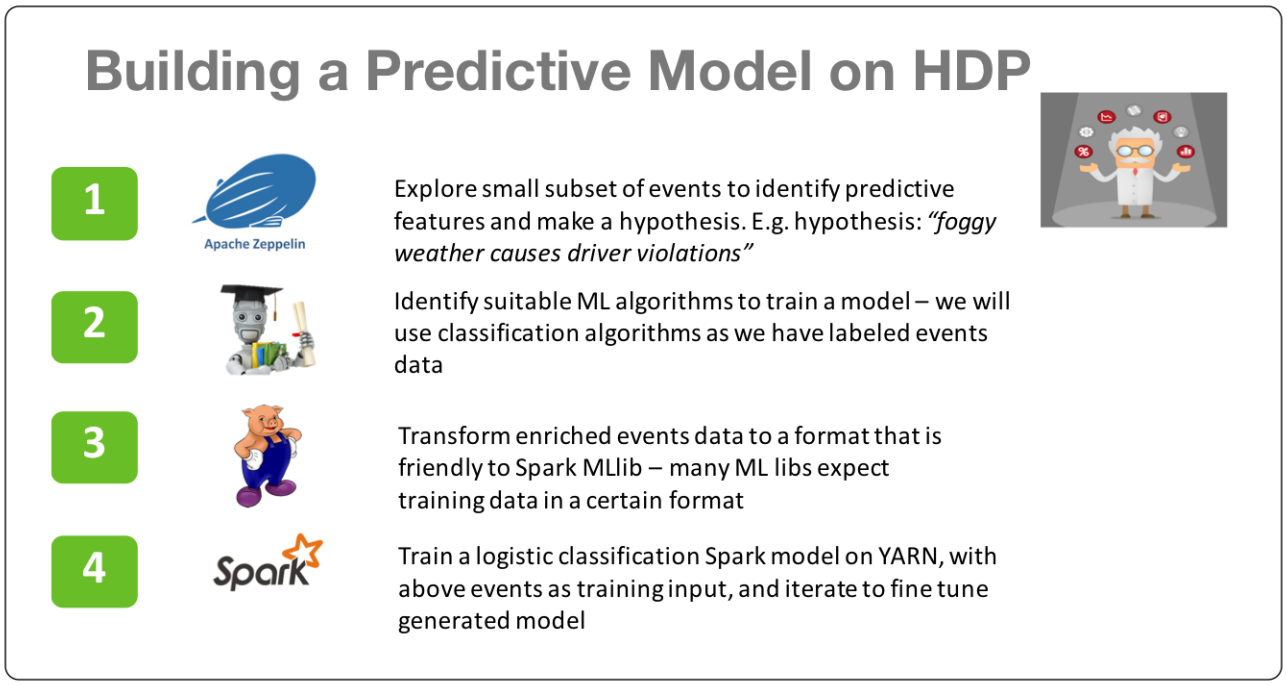
the events, enrich them, and then push them into the truck_events_avro and truck_speed_events_avro Kafka topics. SAM then SAM consumes from those topics.

6. Advanced: Performing Predictive Analytics on the Stream

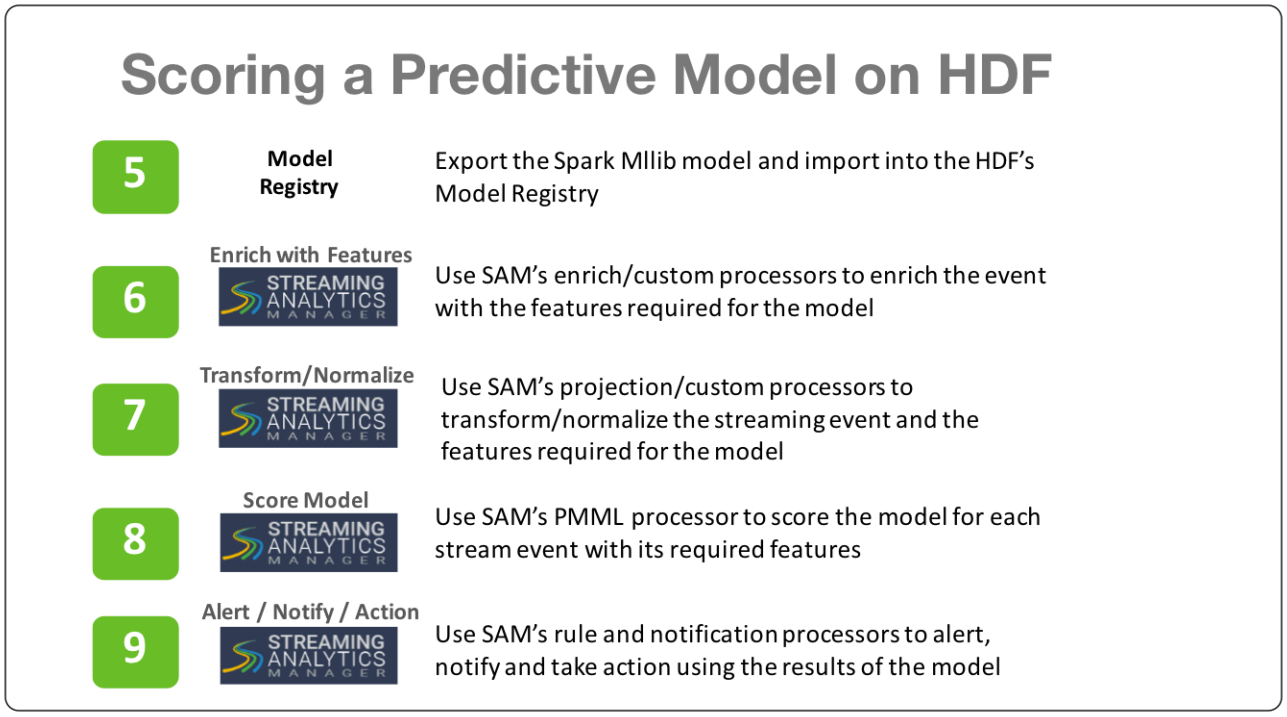
Requirement 10 of this use case states the following:

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 a powerful set of tools for data engineers and scientists to build powerful analytics with data processing engines like Spark Streaming, Hive, and Pig. The following diagram illustrates a typical analytics life cycle in HDP.



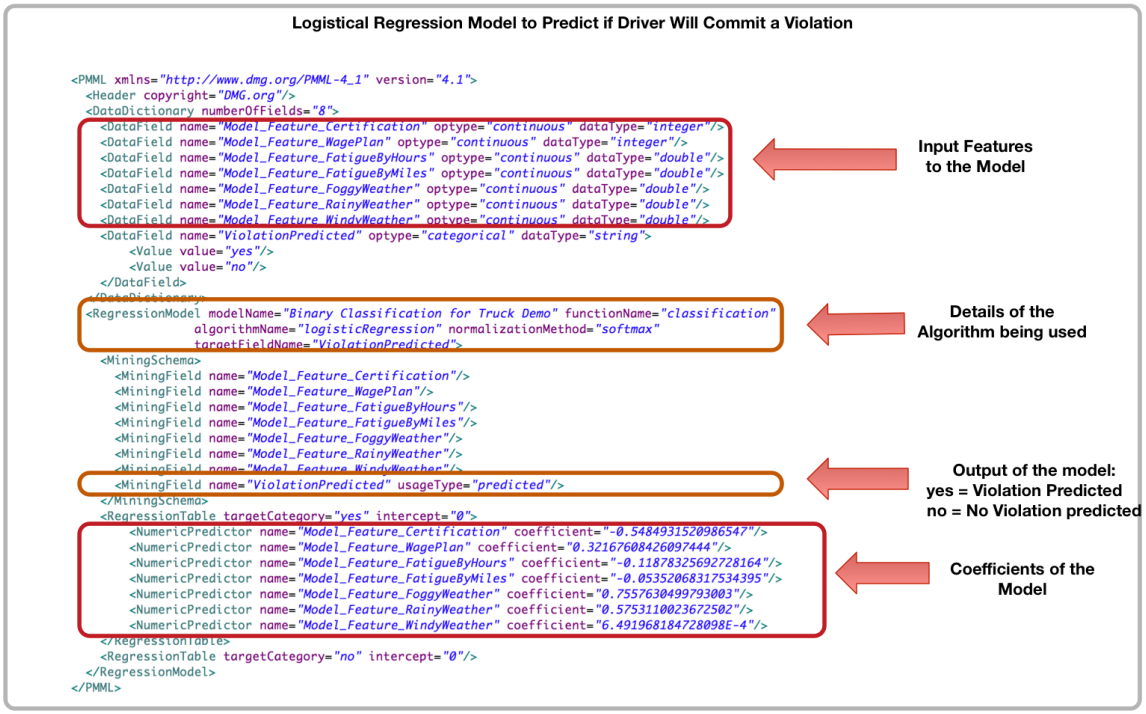
Once the model has been trained and optimized, you can create insights by scoring the model in real-time as events are coming in. The next set of steps in the life cycle score the model in real-time using HDF components.



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

6.1. Logistical Regression Model

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



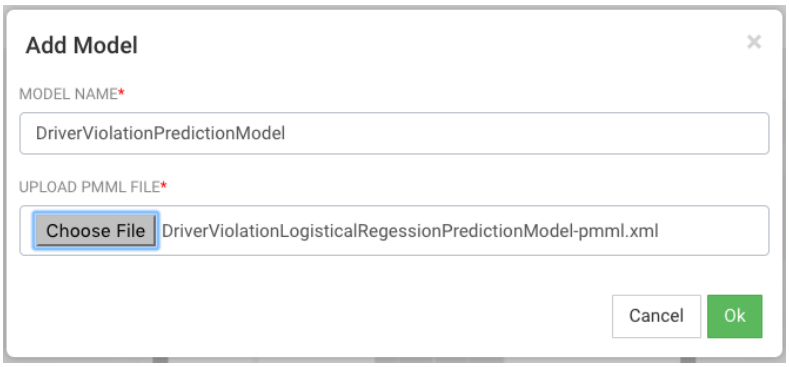
6.2. Export the Model into SAM's Model Registry

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

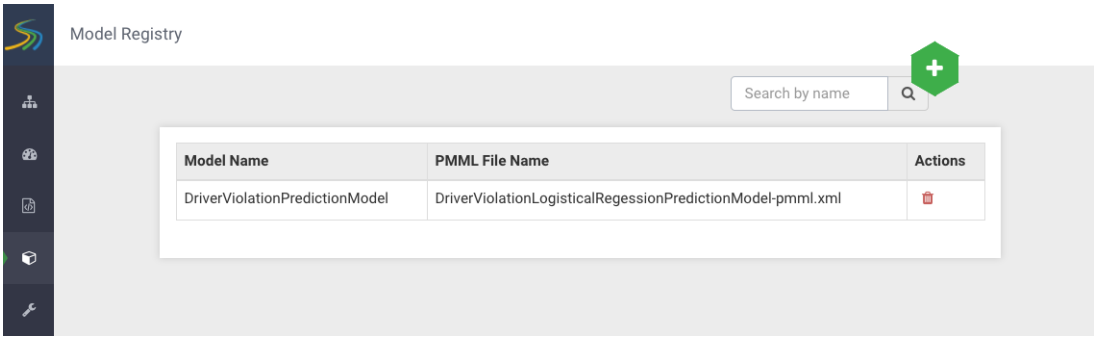
1. Download this PMML model and save it locally with an `.xml` extension.
2. Select the **Model Registry** menu item.
3. Click the + icon.
4. Give your PMML model a name.
5. From **Upload PMML File**, select the PMML file you just downloaded.



6. Click **Ok**.

Result

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



6.3. 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 seven 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_Fatigue ByHours	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_Fatigue ByMiles	The total number of miles driven by the driver in the last week	HBase/Phoenix table called timesheet	Scale by 1000 to improve algorithm performance (e.g.,miles/1000)
Model_Feature_Foggy Weather	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_Rainy Weather	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_Windy Weather	Determines if for the given time and location, if the conditions are windy	API to WeatherService	if (windy) # normalize to 1 else 0

6.4. 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.

6.4.1. Upload Custom UDFs

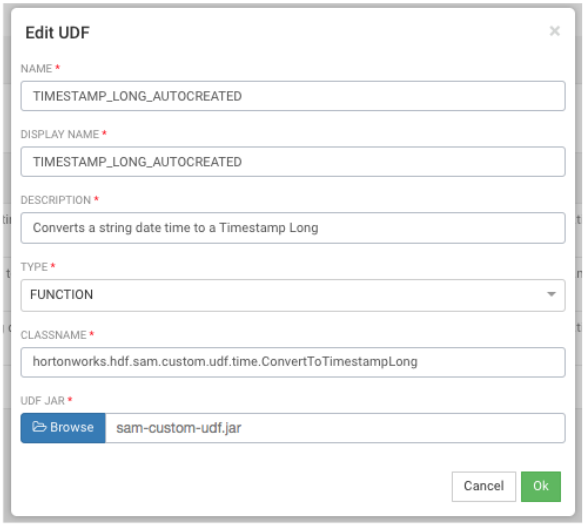
Steps for Uploading the Timestamp_Long UDF

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.

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

3. Configure the UDF with the following values:

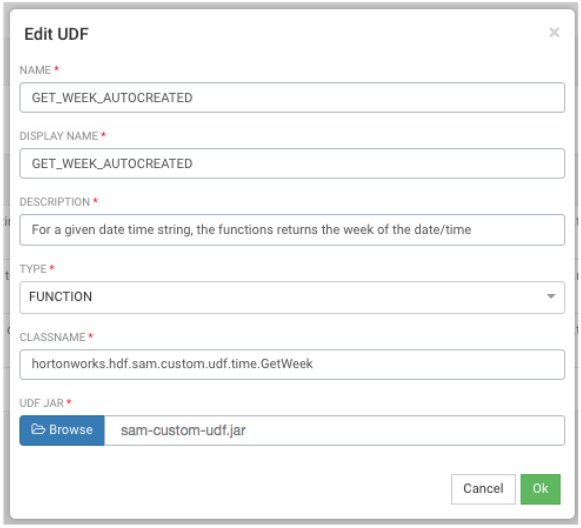


Steps for Configuring the Get_Week UDF

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

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

2. Configure the UDF with the following values:



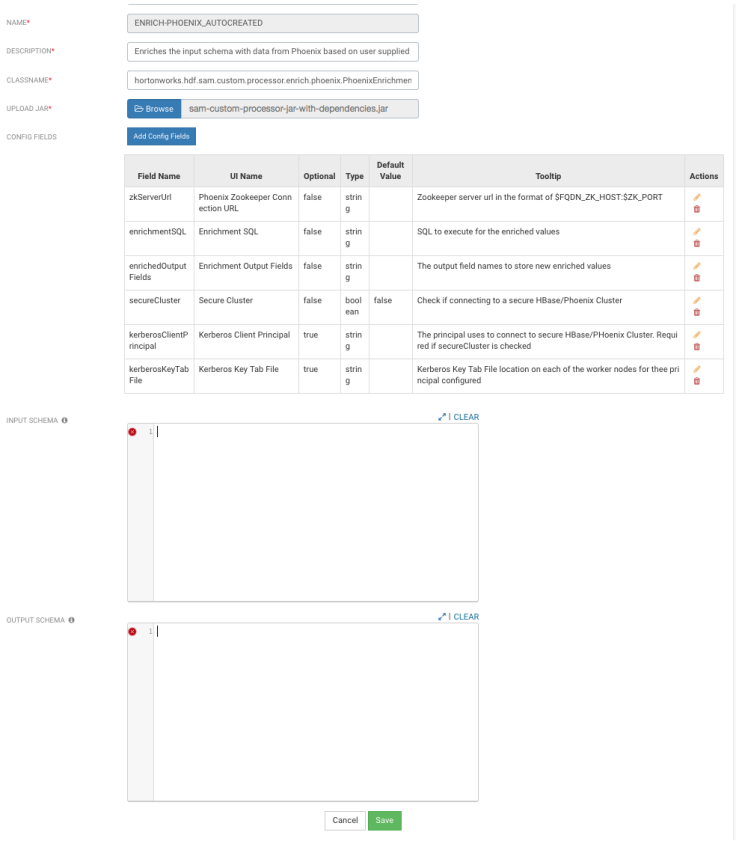
6.4.2. Upload Custom Processors

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.



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/custom-processor/sam-custom-processor-jar-with-dependencies.jar`

Click the **Add Config Fields** button and the following three 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

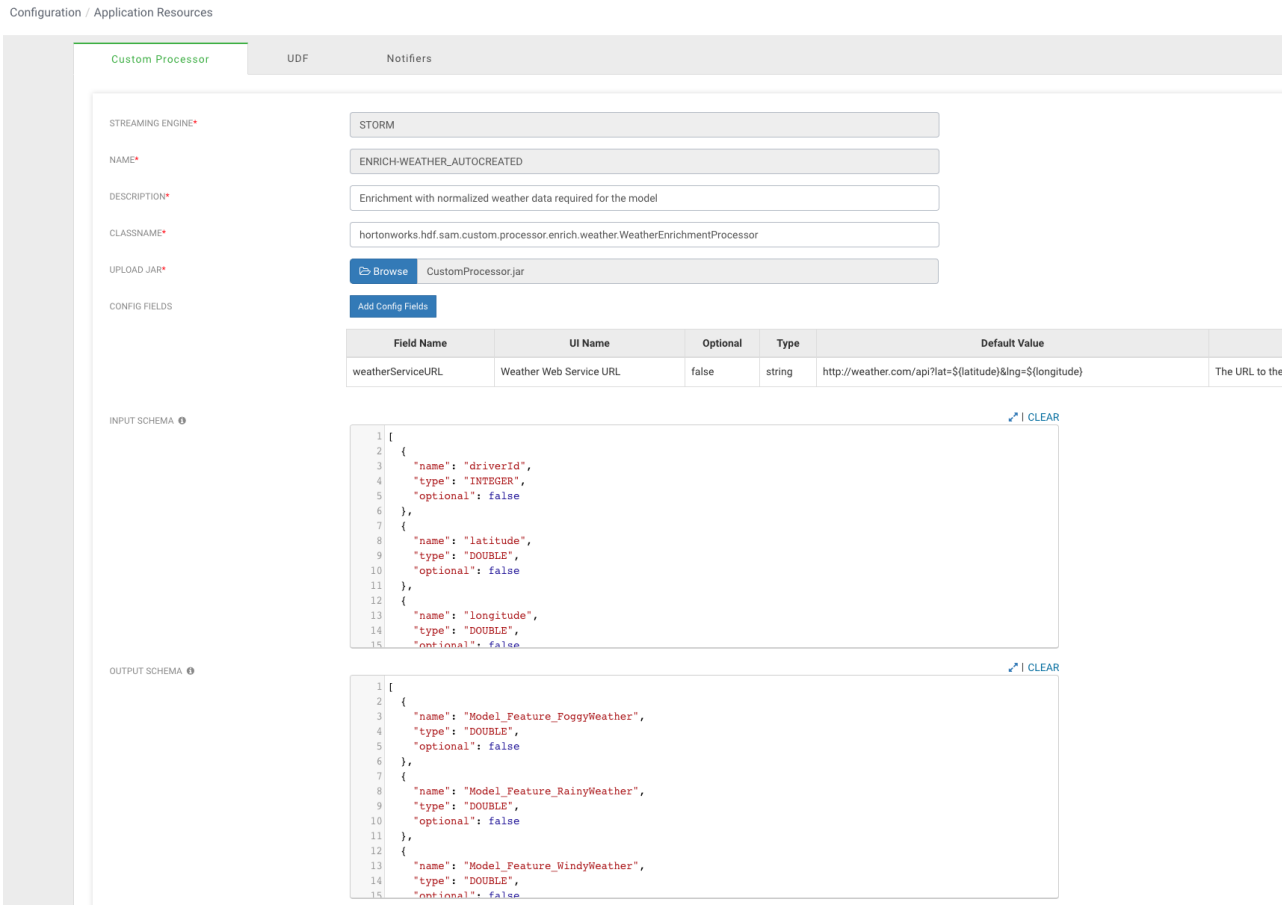
d. **Type** – string

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

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.



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/custom-processor/sam-custom-processor.jar`

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

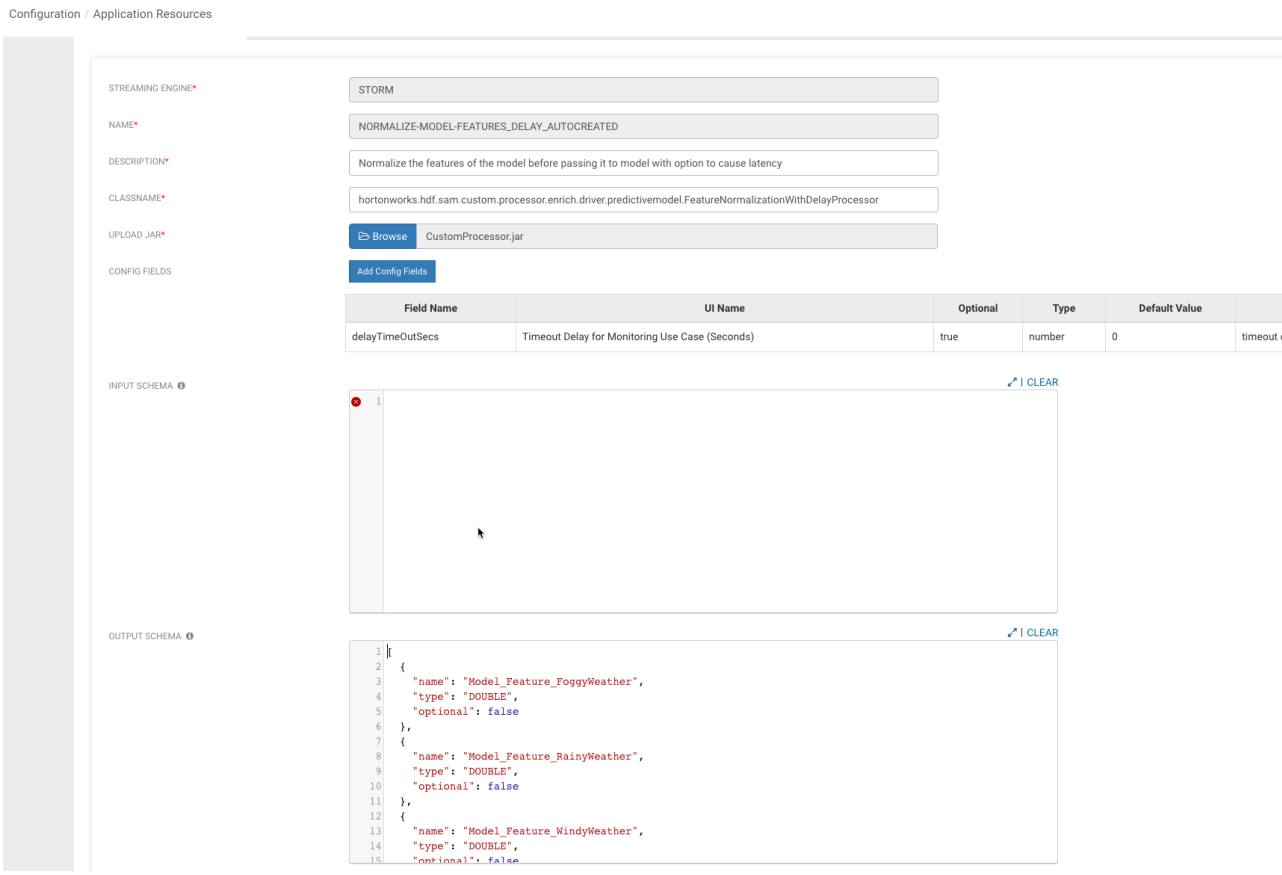
- **Field Name** – weatherServiceURL
- **UI Name** – Weather Web Service URL
- **Optional** – false
- **Type** – string
- **Tooltip** – The URL to the Weather Web Service

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 normalizes the enriched fields to a format that the model is expecting.
2. Configure the processor with the following values:



NORMALIZE-MODEL-FEATURES Configuration Values

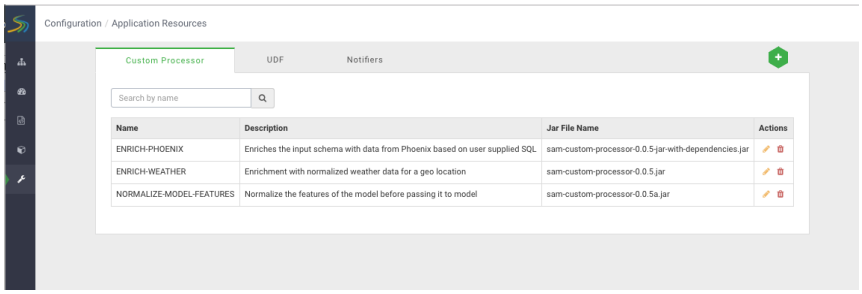
- **Streaming Engine** – Storm
- **Name** – NORMALIZE-MODEL-FEATURES
- **Description** – Normalize the features of the model before passing it to model
- **ClassName** – hortonworks.hdf.sam.custom.processor.enrich.driver.predictivemodel.FeatureNormalizationProcessor
- **Upload Jar** – The jar for this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

Input and Output Schema for NORMALIZE-MODEL-FEATURES

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

Result

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



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

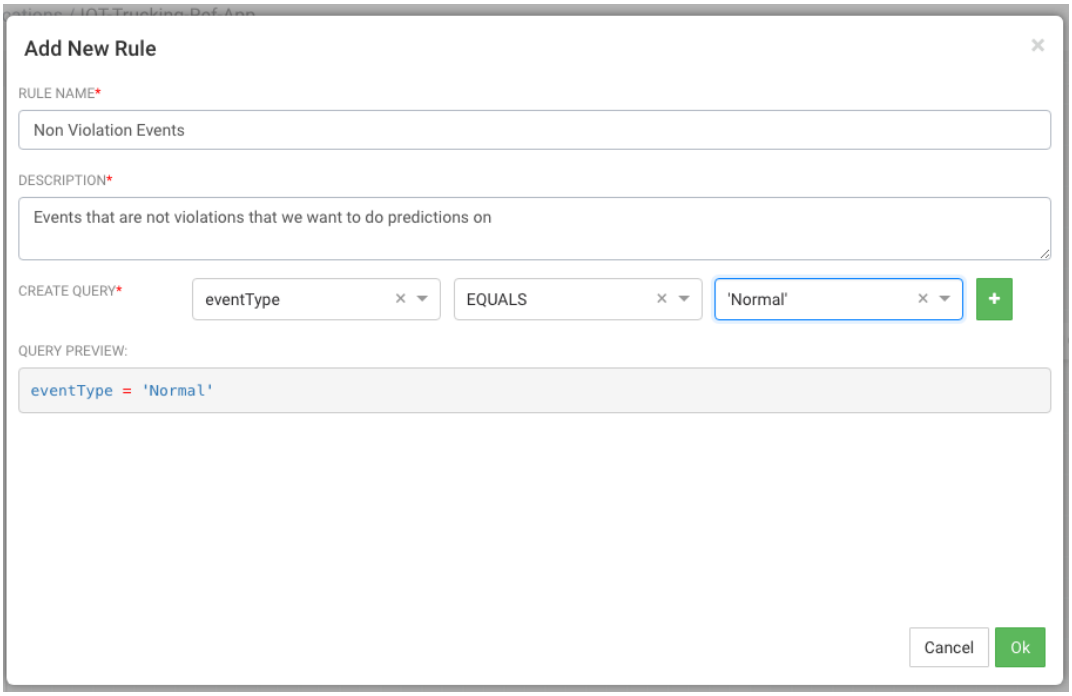
6.5. Scoring the Model in the Stream using a Streaming Split Join Pattern

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.

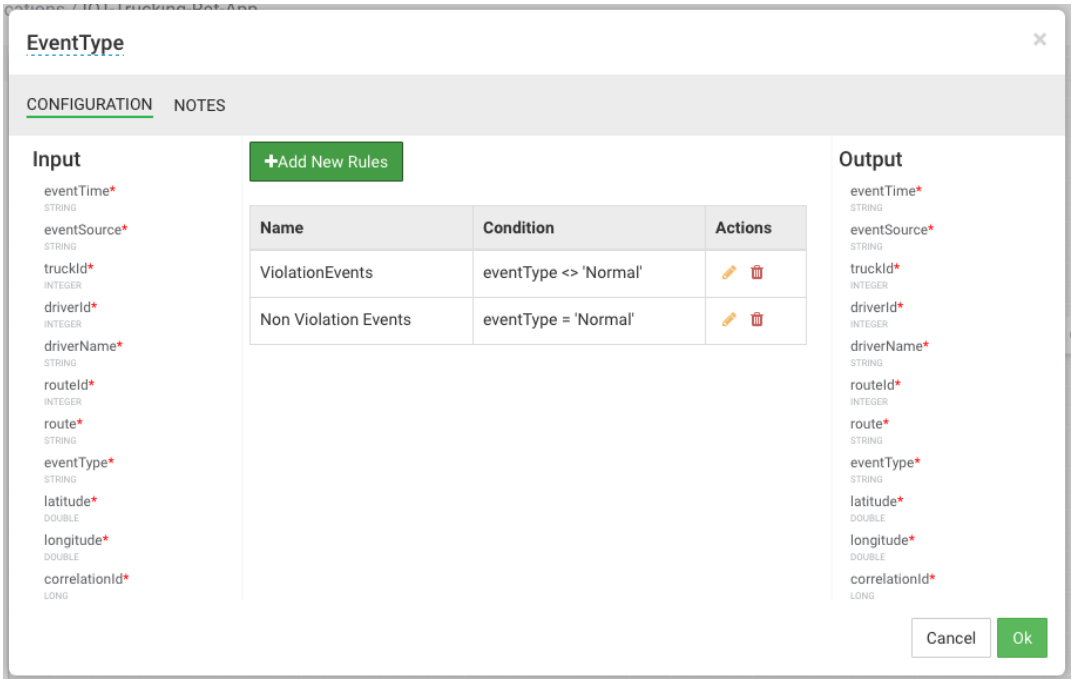
Steps

1. Click into 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:



Result

Your new rule is added to the Event Type processor.



6.6. Streaming Split Join Pattern

About This Task

Your 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.

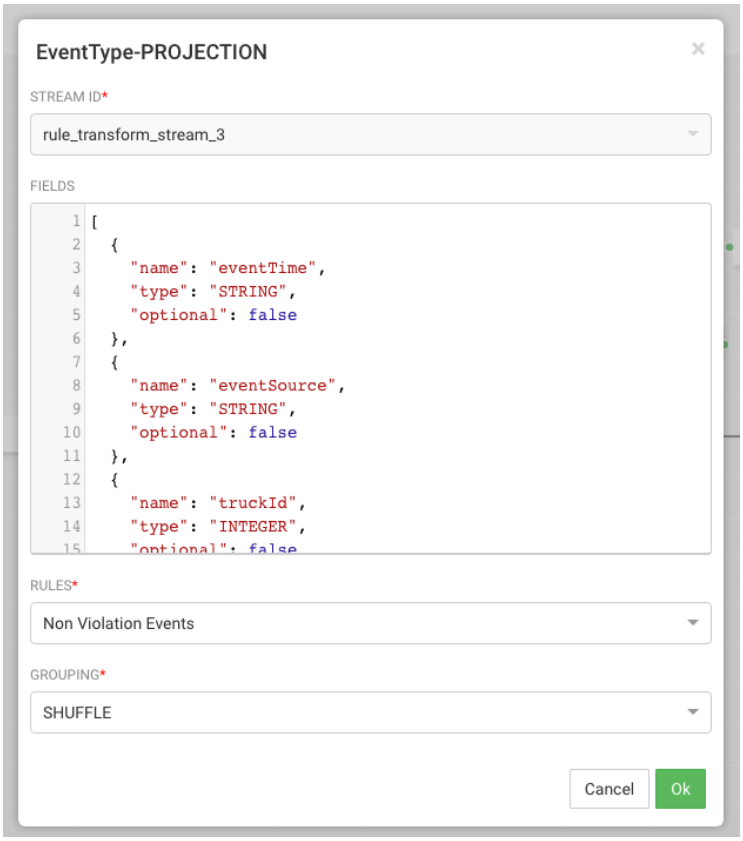
To do this, use the split join pattern to split the stream into three, perform the enrichment in parallel, and then re-join the three streams.

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.

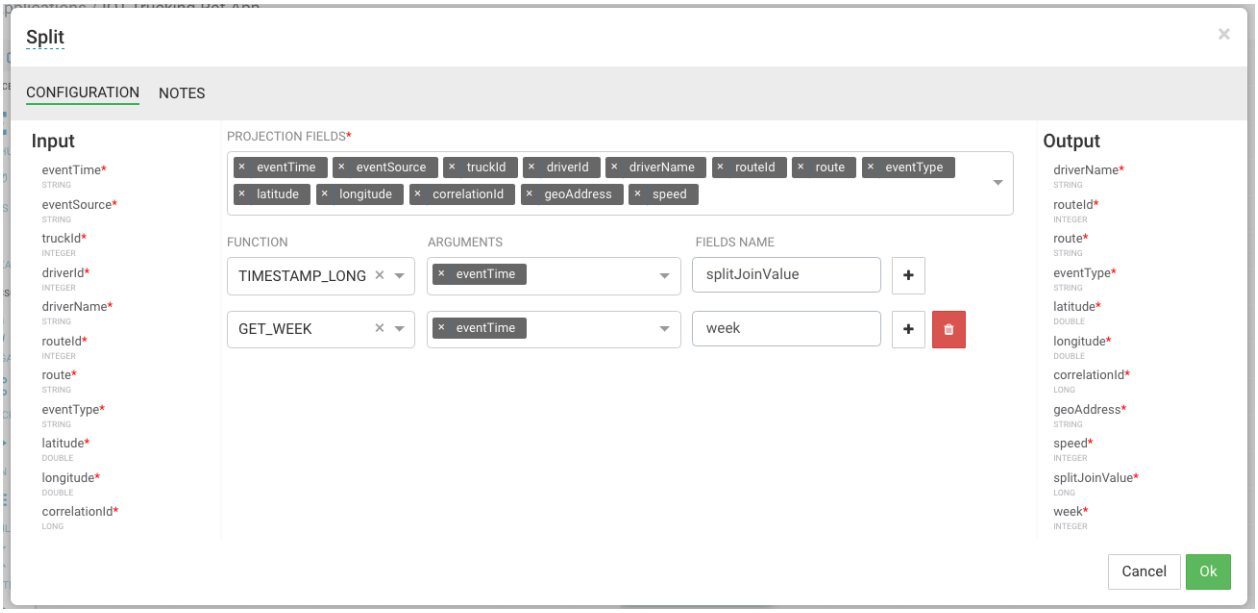
   To do this, drag the projection processor to the canvas and create a connection from the EventType rule processor to this projection processor.

   When configuring the connection, select the Non Violation Events Rule which tells SAM to only send non-violation events to this project processor.



2. Configure the projection processor to create the split join key called `splitJoinValue` using the custom UDF you uploaded earlier called "TIMESTAMP_LONG".

   You 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 with the following parameters:



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

1. With the split join key created, you can split the stream into three to perform the enrichments in parallel.

   To do the first split to enrichment the wage and certification status of driver, drag the "ENRICH-PHOENIX" processor to the canvas and connect it from the Split project processor.

2. Configure the enrich processor with the following parameters:

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

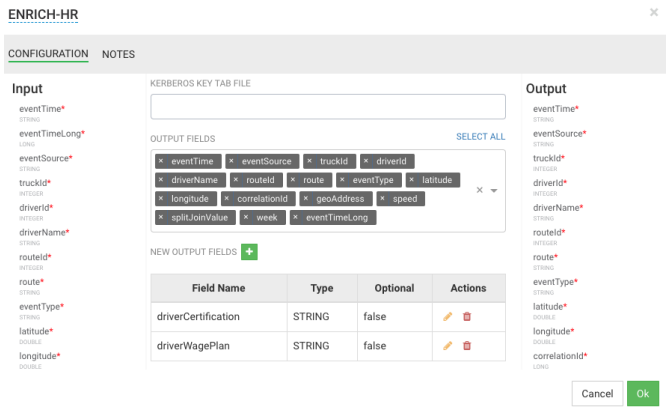
   b. ENRICHMENT OUTPUT FIELDS: driverCertification, driverWagePlan

   c. SECURE CLUSTER: false

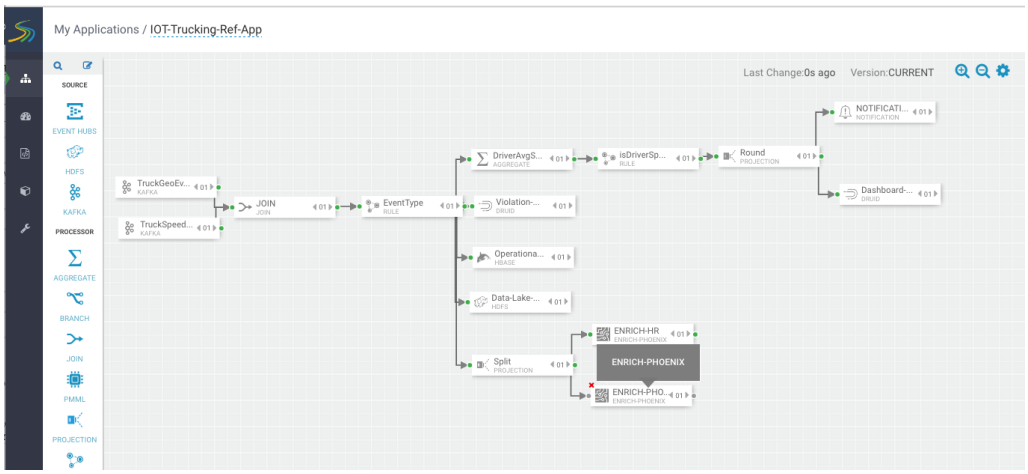
   d. OUTPUT FIELDS: Click **Select All**.

   e. NEW OUTPUT FIELDS: Add new output fields for the two enriched values: `driverCertification` and `driverWagePlan`.

   After this processor executes, the output schema will have two fields populated called driverCertification and driverWagePlan.



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



4. Configure the enrich processor with the following parameters:

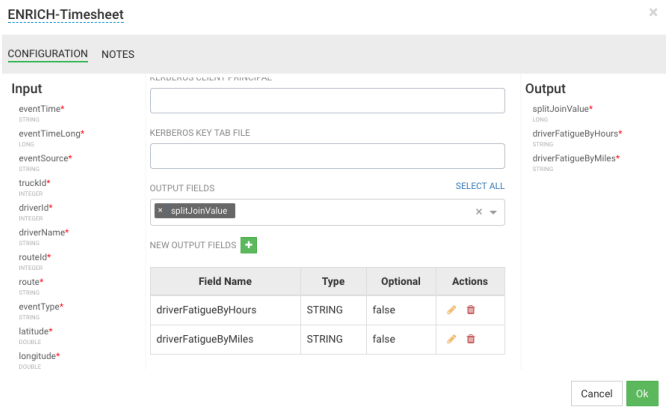
   a. ENRICHEMNT SQL: select hours_logged, miles_logged from timesheet where driverid=${driverId} and week=${week}

   b. ENRICHMENT OUTPUT FIELDS: driverFatigueByHours, driverFatigueByMiles

   c. SECURE CLUSTER: false

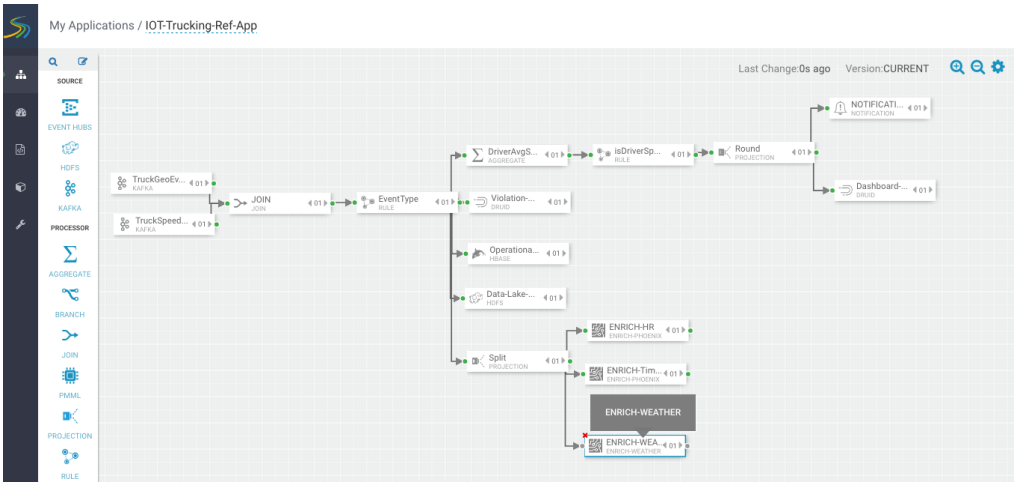
   d. OUTPUT FIELDS: Select the `splitJoinValue` field.

   e. NEW OUTPUT FIELDS: Add new output fields for the two enriched values `driverFatigueByHours` and `driverFatigueByMiles`.



After this processor executes, the output schema will have two fields populated called driverFatigueByHours and driverFatigueByMiles.

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

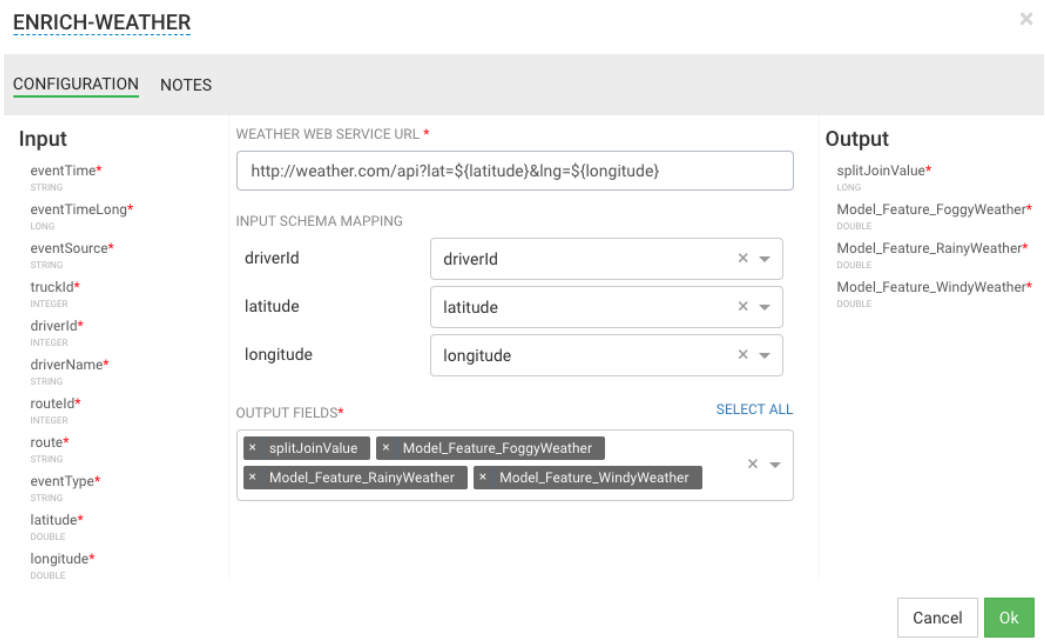


6. Configure the weather process with the following parameters (currently the weather processor is just a stub that generates random normalized weather info).

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

   b. INPUT SCHEMA MAPPINGS: Leave defaults

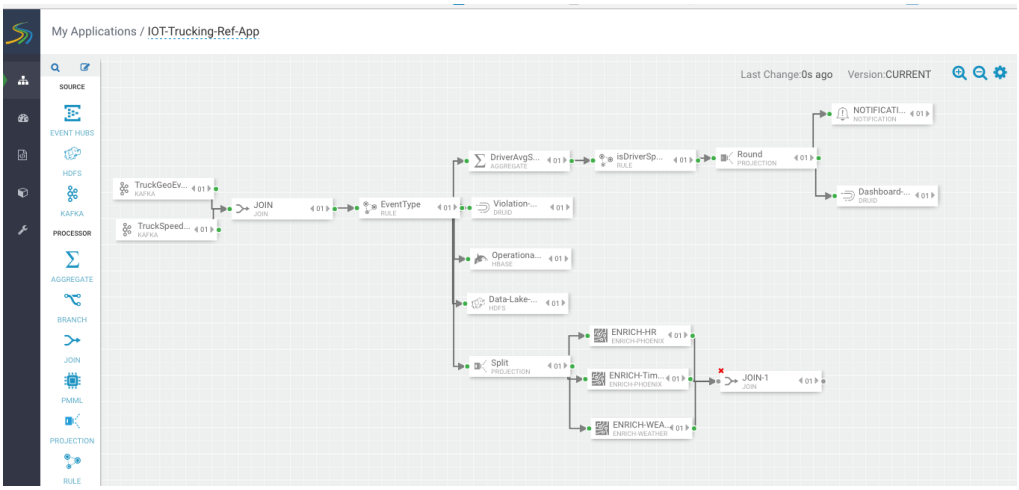
   c. OUTPUT FIELDS: Select the splitJoinValue and the three model enriched features



After this processor executes, the output schema will have three fields populated called Model_Feature_FoggyWeather, Model_Feature_RainyWeather, and Model_Feature_WindyWeather.

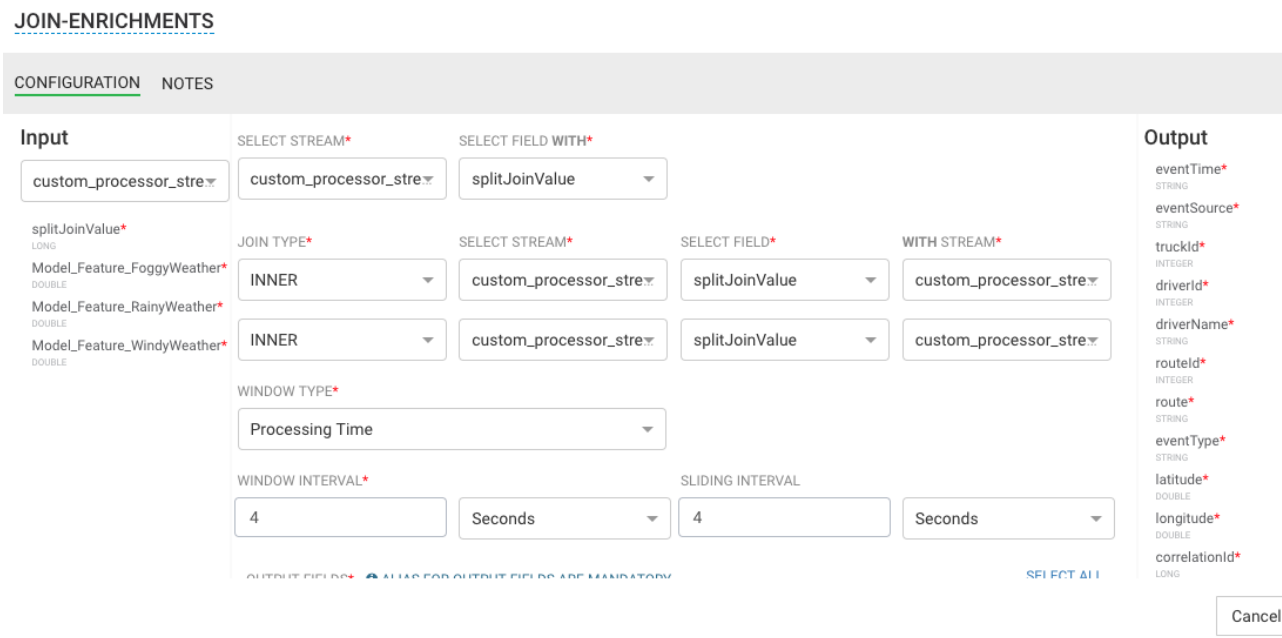
Steps for Rejoining the Three Enriched Streams

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



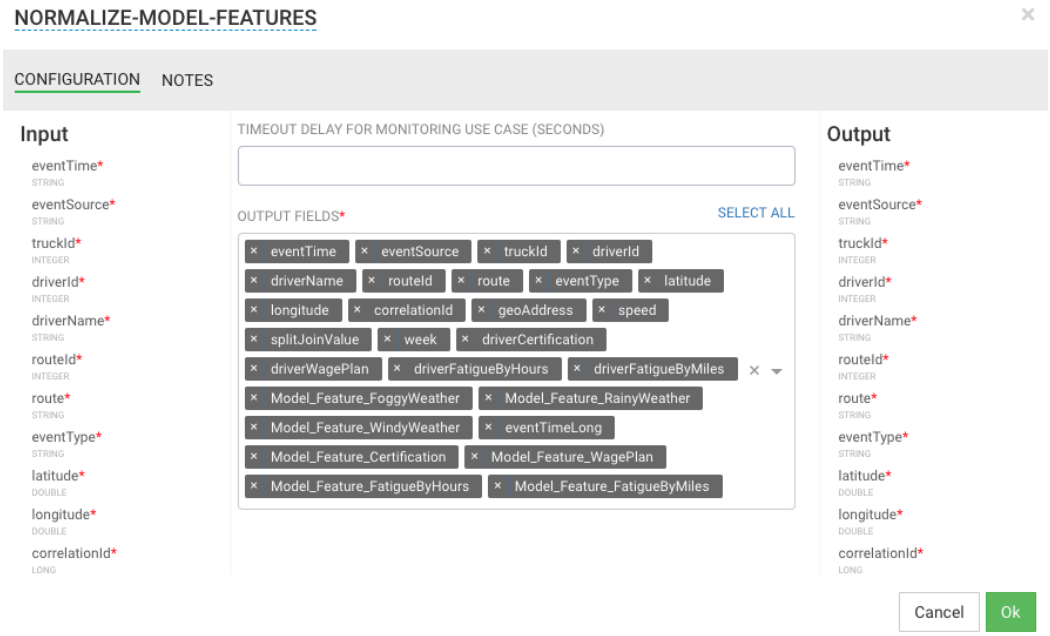
2. Configure the join processor like the following where you use the joinSplitValue to join all three streams.

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



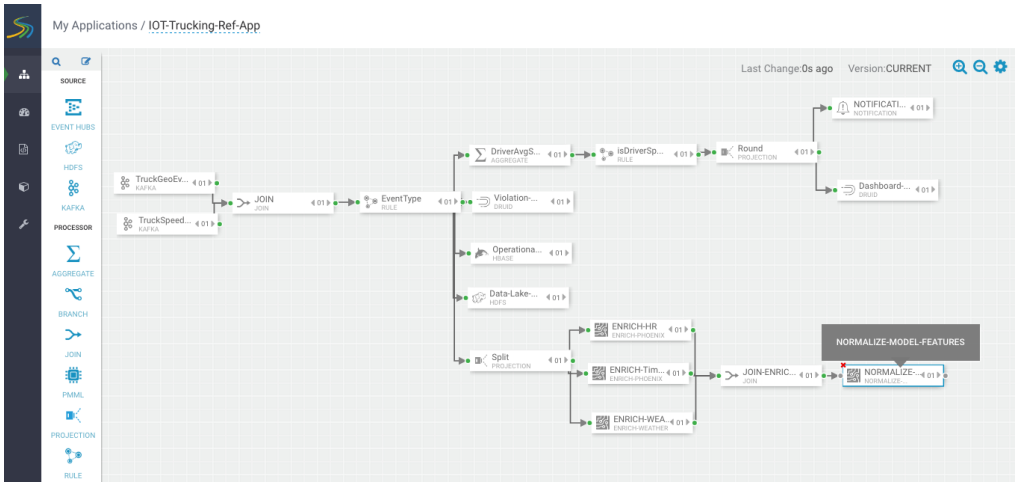
3. Now that you have joined three enriched streams, normalize the data into the format that the model expects by dragging the "NORMALIZE-MODEL-FEATURES" custom processor that you added to the canvas.

   For the output fields, select all the fields and leave the mapping as defaults.



Result

Your flow looks similar to the following.



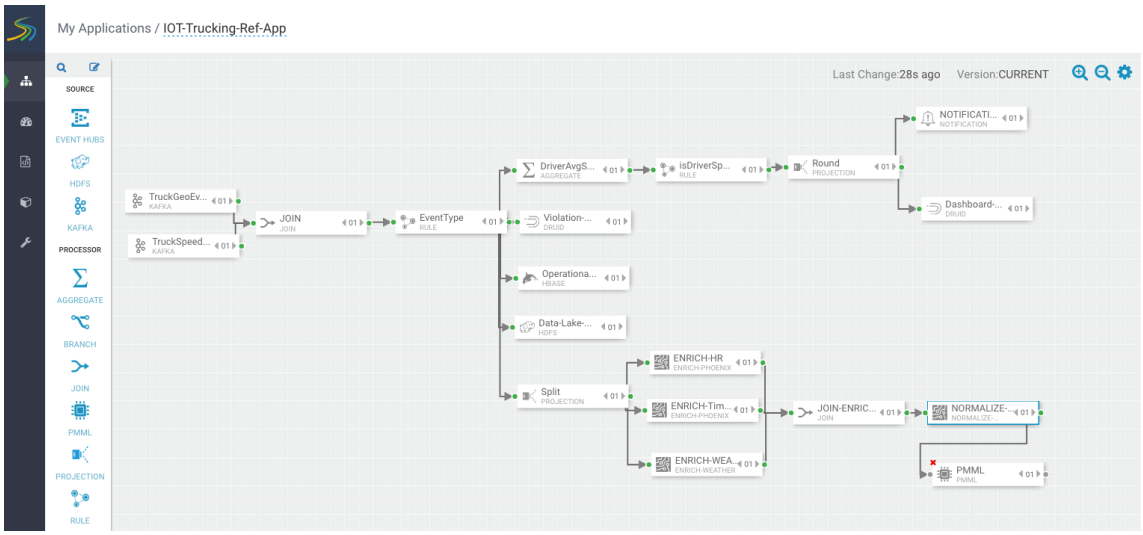
6.7. Score the Model Using the PMML Processor and Alert

About This Task

Now you are ready to score the logistical regression model.

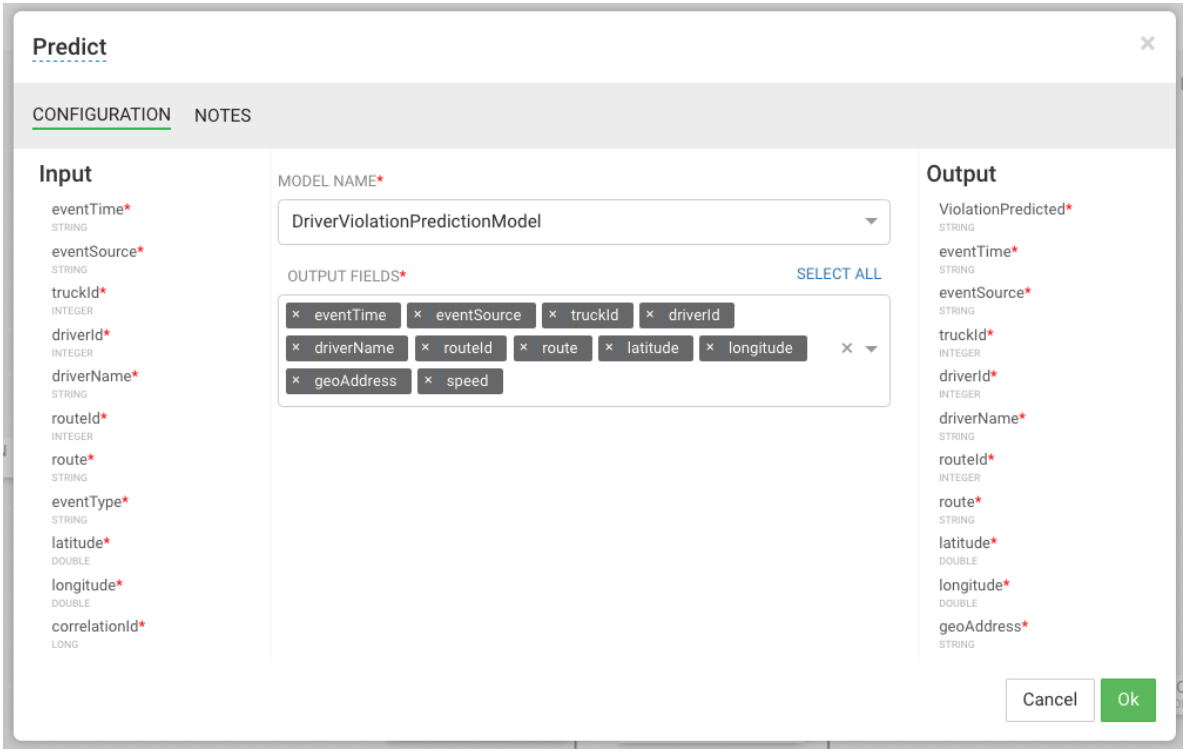
Steps

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

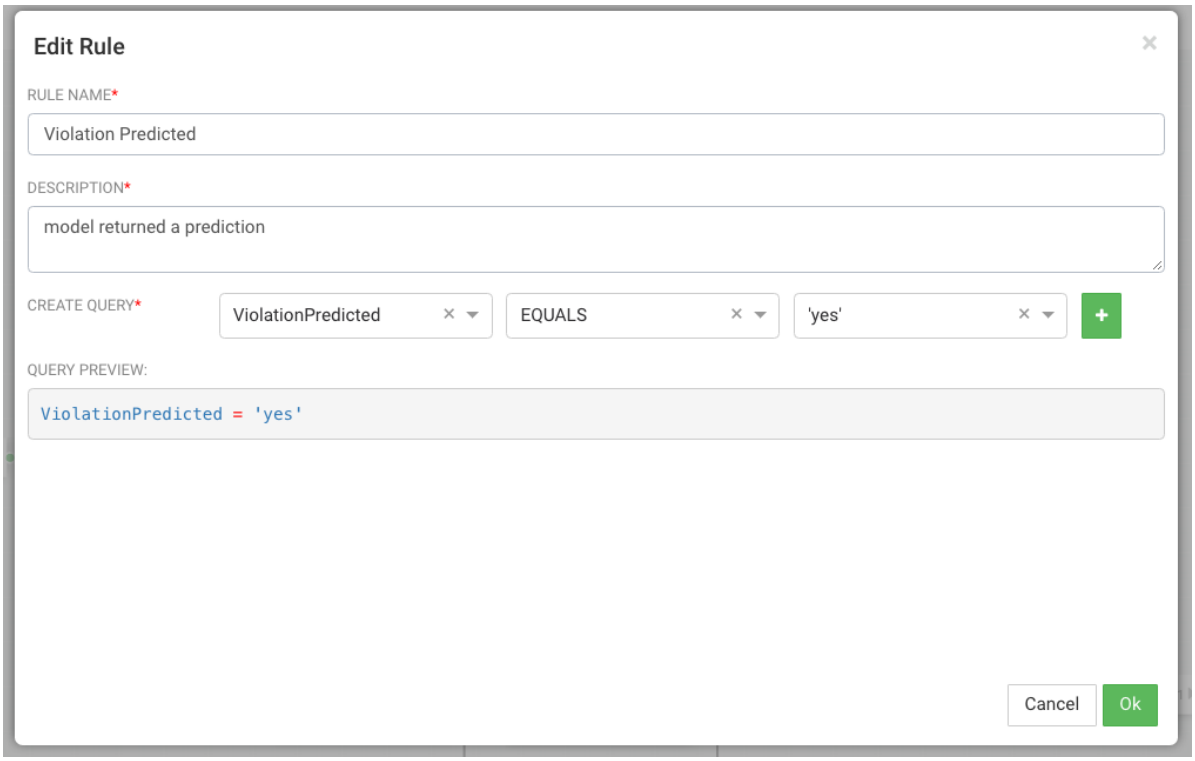


2. Configure the PMML processor like the following by selecting the DriverViolationPredictionModel that you uploaded earlier to the **Model Registry**.

   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.



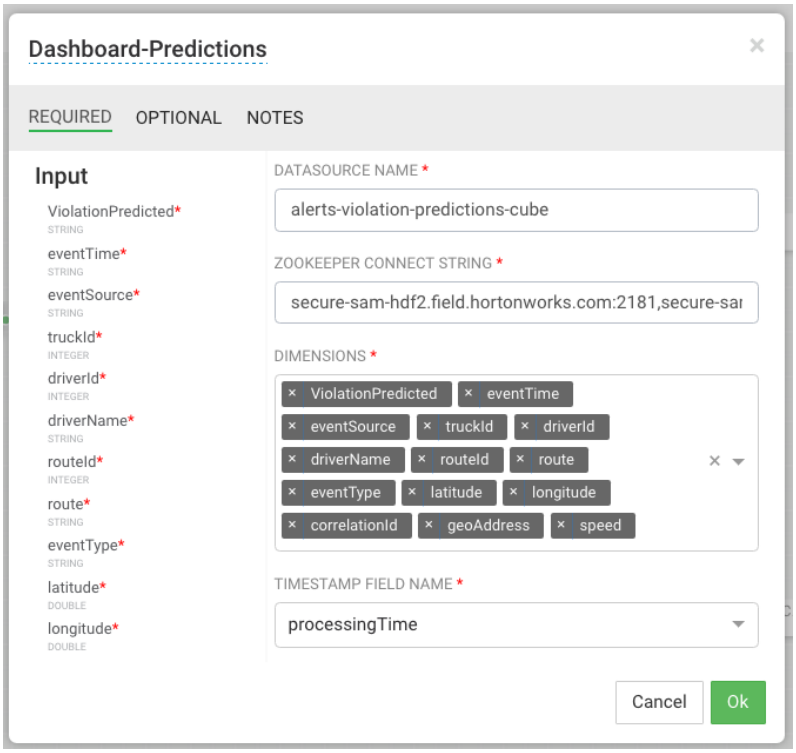
3. Determine if the model predicted if the driver will commit a violation by dragging a rule processor to the canvas and configuring a rule like the following:



4. If a violation is predicted, send it to a Druid to display on a dashboard.

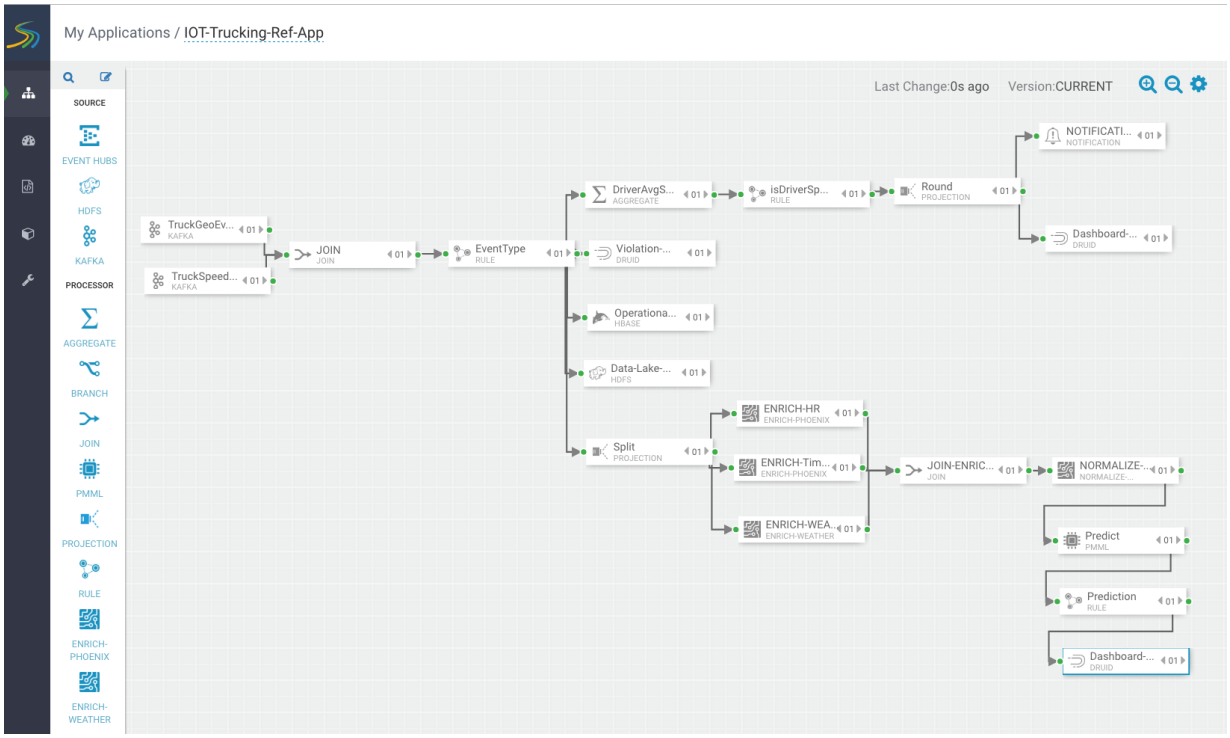
Drag the Druid processor to canvas and configure.

5. Stream the events into a cube called **alerts-violation-predictions-cube**.



Result

The final flow looks like the following:


My Applications / IOT-Trucking-Ref-App
Last Change:0s ago
Version:CURRENT
SOURCE
EVENT HUBS
HDFS
KAFKA
PROCESSOR
AGGREGATE
BRANCH
JOIN
PMML
PROJECTION
RULE
ENRICH-PHOENIX
ENRICH-WEATHER
TruckGeoEv...
TruckSpeed...
JOIN
EventType
DriverAvgS...
Violation-...
Operationa...
Data-Lake-...
Split
isDriverSp...
Round
NOTIFICATI...
Dashboard-...
ENRICH-HR
ENRICH-Tim...
ENRICH-WEA...
JOIN-ENRIC...
NORMALIZE-...
Predict
Prediction
Dashboard-...

7. 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 associated with a data source (for example, Druid cube).

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

Note

When a SAM app streams data to a new cube using the Druid processor, it will take 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.

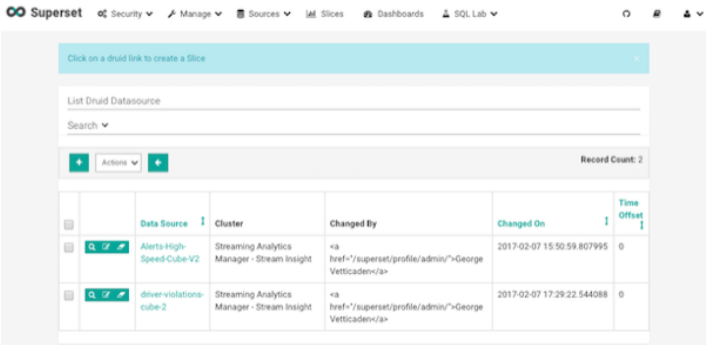
7.1. Creating Insight Slices

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

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

2. Click + to create a new Slice.

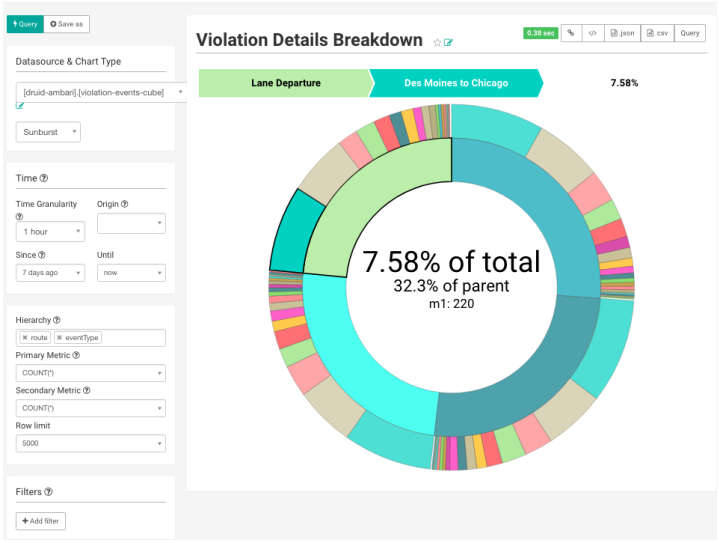
3. Select the Druid Data Source that you want to use for the new visualization:



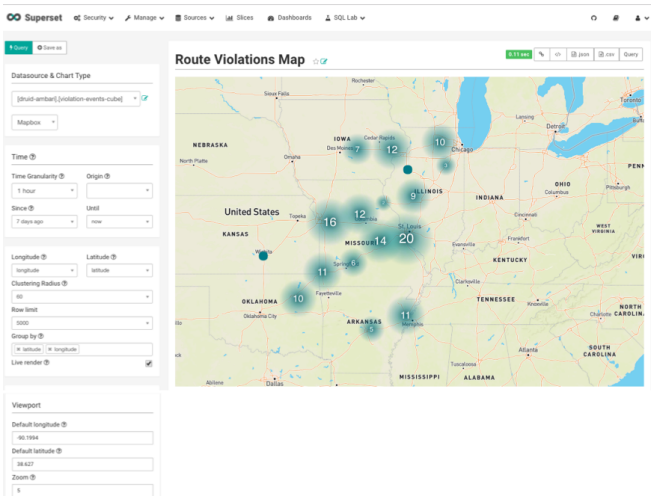
4. Select a Chart Type from the menu.

   The following example creates a "Sunburst" visualization of rolling up multiple dimensions like route, eventType, and driver info..

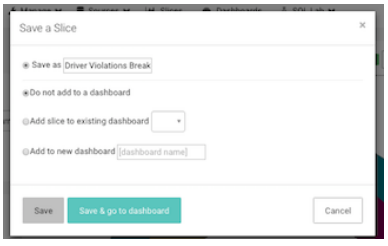
   Configure the chart and click **Execute Query**



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



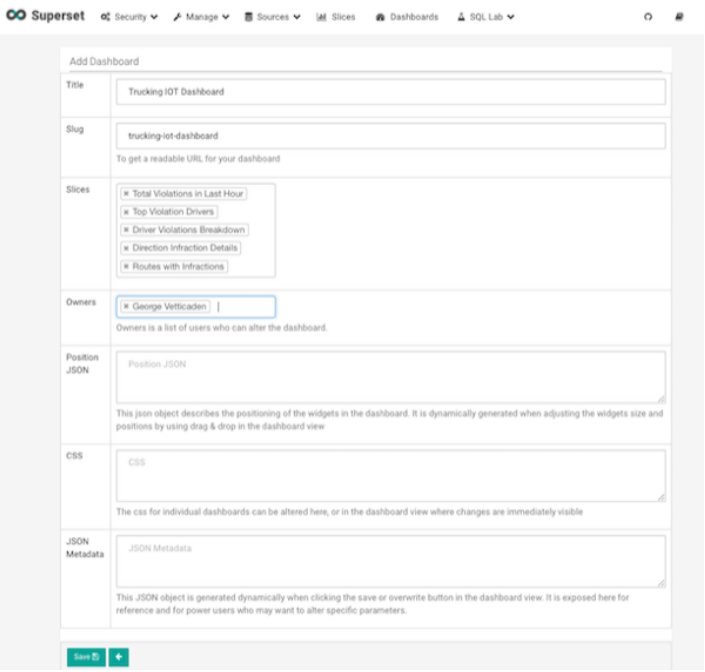
6. To save the slice, specify a name and name and click **Save**.



7.2. 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.



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

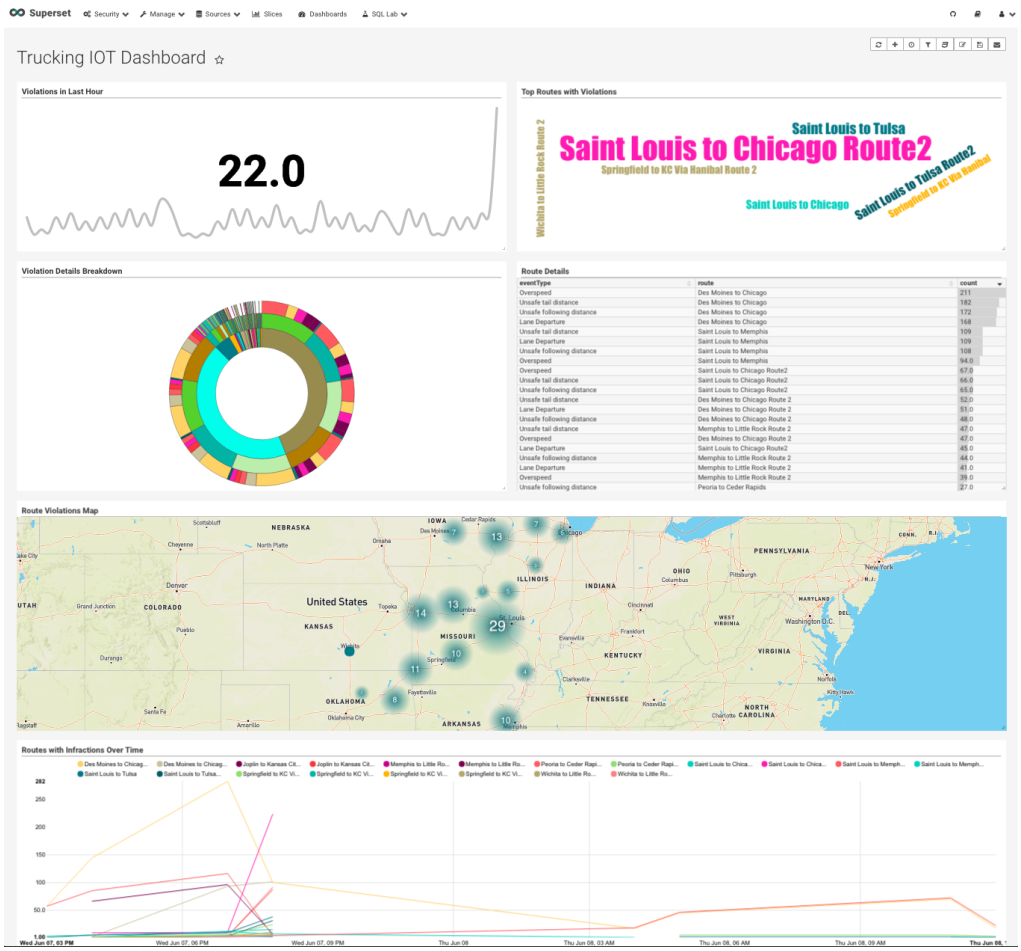
7.2.1. Dashboards for the Trucking IOT App

The IOT Trucking application that we implemented using the Stream Builder streams violation events, alerts, and predictions into three cubes:

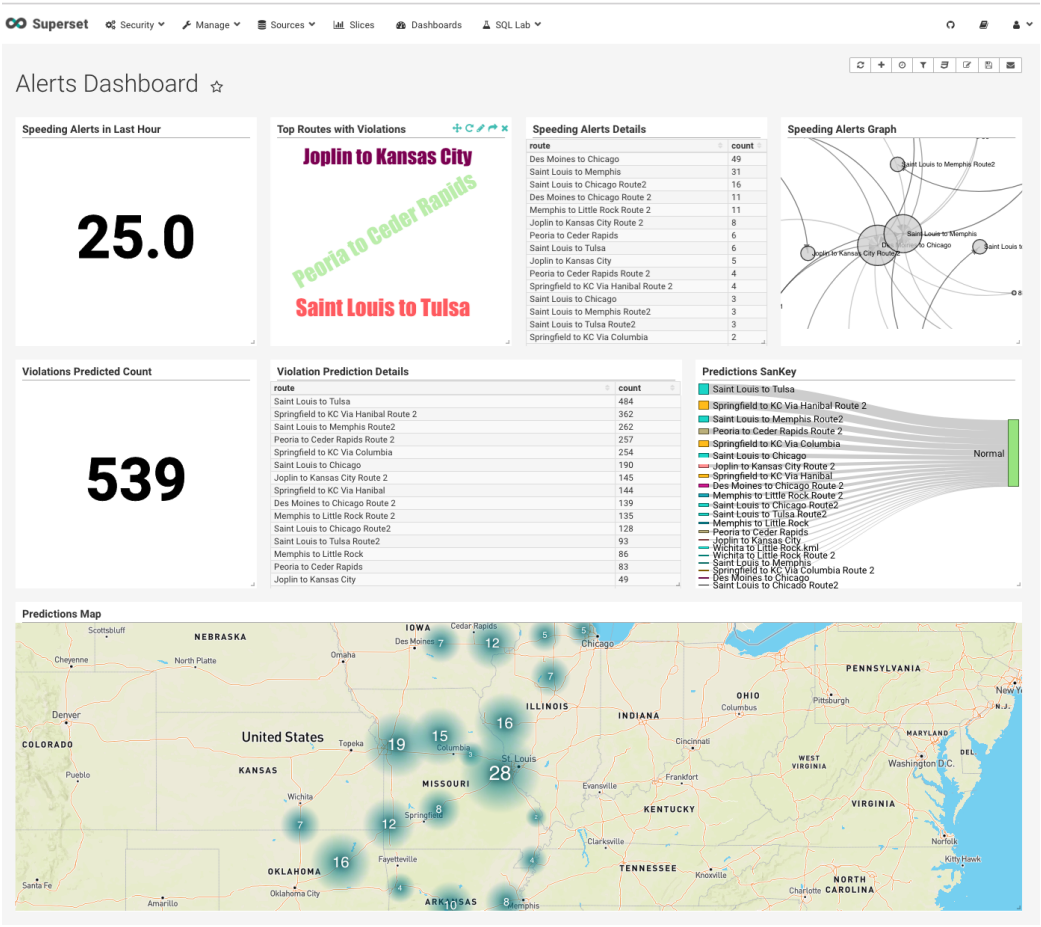
- violation-events-cube
- alerts-speeding-drivers-cube
- alerts-violation-predictions-cube

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

IoT Dashboard



Alerts Dashboard


Superset
Security
Manage
Sources
Slices
Dashboards
SQL Lab
Alerts Dashboard
Speeding Alerts in Last Hour
25.0
Top Routes with Violations
Joplin to Kansas City
Peoria to Ceder Rapids
Saint Louis to Tulsa
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 Chicado Route2
Normal
Predictions Map
NEBRASKA
IOWA
ILLINOIS
INDIANA
OHIO
PENNSYLVANIA
COLORADO
KANSAS
MISSOURI
KENTUCKY
WEST VIRGINIA
VIRGINIA
MARYLAND
OKLAHOMA
TENNESSEE
NORTH CAROLINA
United States
St. Louis
Chicago
Denver
Topeka
Columbia
Springfield
Wichita
Fayetteville
Oklahoma City
Memphis
Evansville
Frankfort
Cincinnati
Columbus
Pittsburgh
Washington D.C.
Norfolk
Kitty Hawk
Charlotte
Knoxville
Clarksville
Santa Fe
Amarillo
Pueblo
Cheyenne
North Platte
Omaha
Des Moines
Cedar Rapids
Scottsbluff
Saint Louis to Memphis Route2
Saint Louis to Memphis
Joplin to Kansas City Route2
Des Moines to Chicago
Saint Louis to Chicago Route2

8. SAM Test Mode

In a typical SDLC (Software development lifecycle), you want to test the streaming analytics app locally before deploying the SAM app to a cluster. SAM's "Test Mode" allows you to test the app locally using test data for the sources. SAM's Test Mode allows you to do the following:

- Create a Named Test Case
- Mock out the sources of the app and configure test data for each test source. SAM validates the test data using the configured Schema in the Schema Registry for each source
- Execute the Test Case and visually see how the data looks like at each component/ processor in the app as flows across your application.
- Download the output of the test which represents the state of the data at each processor and sink.

In the following sections, we will walk through creating Test Cases for different test scenarios for the reference app. If you ran the test utility, these 4 test cases will already be created for you.

8.1. Four Test Cases using SAM's Test Mode

8.1.1. Test Case 1: Testing Normal Event with No Violation Prediction

The Assertions of this test case are the following:

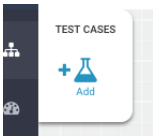
- Assertion 1: Validate test data for geo steam and speed stream that are non violations
- Assertion 2: Validate the Join of data between geo stream and speed stream
- Assertion 3: Validate that the filter "EventType" detects that this is a "Non Violation Event"
- Assertion 4: Validate the joined event gets split into three events by the "Split" projection.
- Assertion 5: Validate that the three enrichments are applied: weather enrichments, timesheet enrichment and HR enrichment.
- Assertion 6: Validate the three enrichment streams are joined into a single stream.
- Assertion 7: Validate that data after normalization for the model
- Assertion 8: Validate the output of the Prediction model is that no violation is predicted

- Assertion 9: Validate the filter “Prediction” detects that it is non violation.

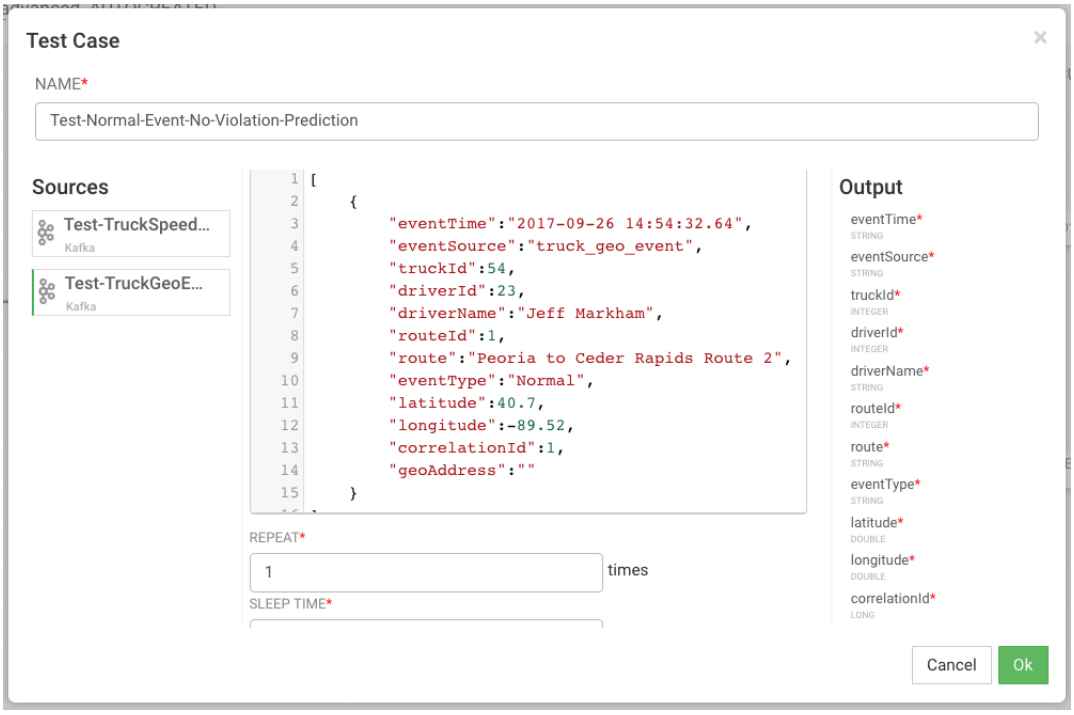
Follow the below steps to create this test case for the reference app in Edit Mode. (If you ran the test utility, these 4 test cases will already be created for you.)

1. Click “TEST” Mode

2. Click Add Test Case

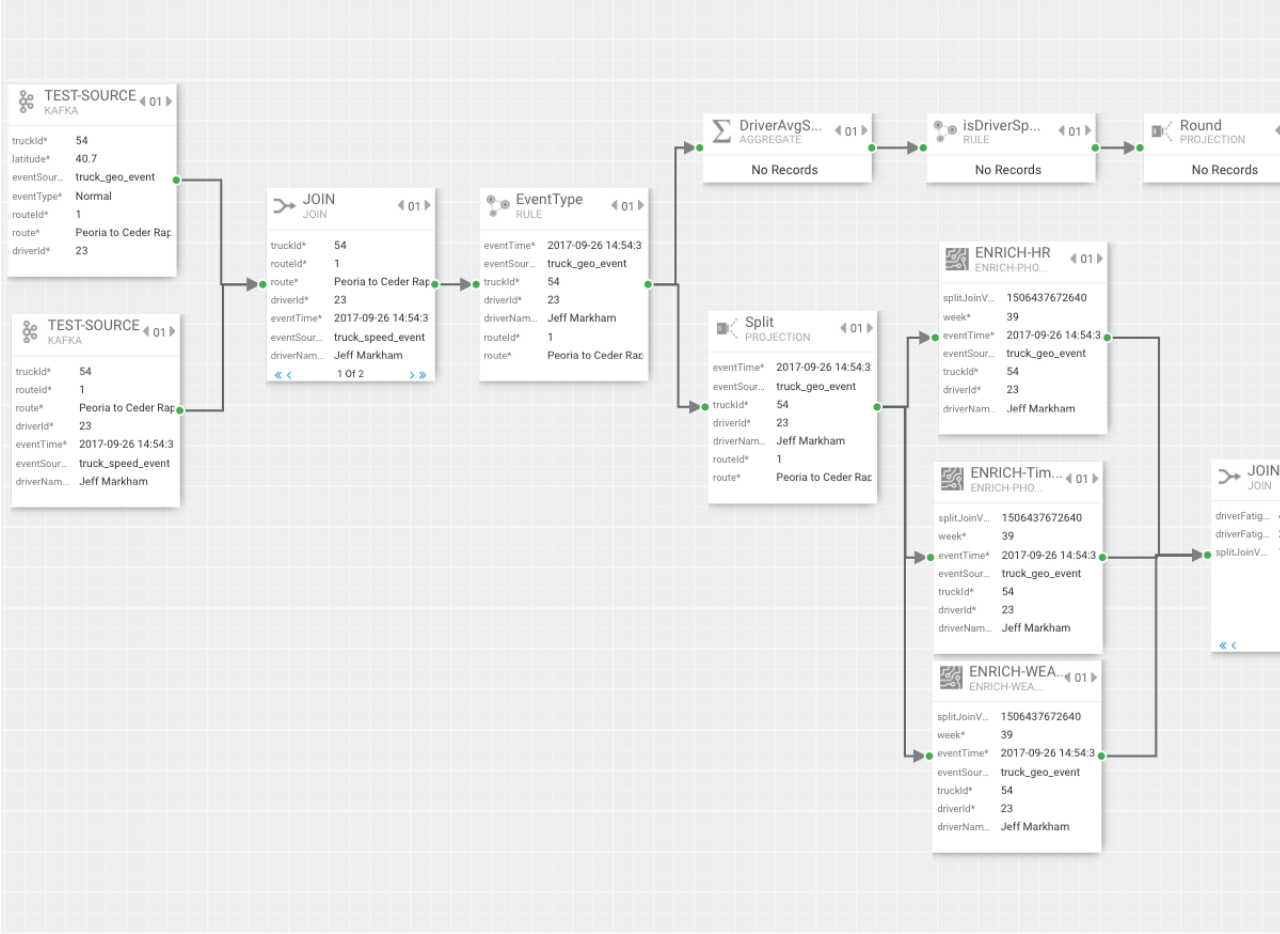


3. Provide Test Case details. Provide a name for test case, test data for TruckGeoEvent and test data for TruckSpeedEvent.

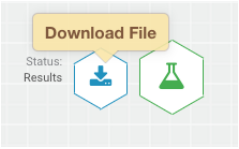


4. Execute the Test Case.

5. You should see the result of the test case as the following.



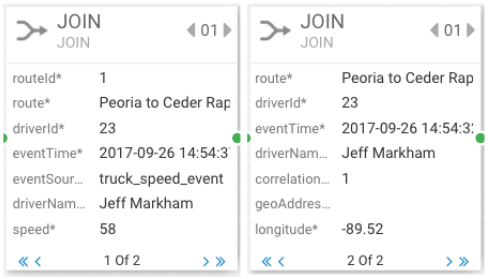
6. Download the test case results.



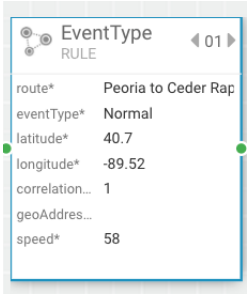
8.1.2. Analyzing Test Case 1 Results

1. The key to reading the test case results is to keep in mind that when you look at the results of the component, you are viewing the input into that component.

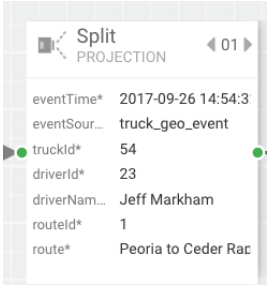
   - Assertion 1 is to Validate test data for geo steam and speed stream that are non violations. For this assertion, you would look at the downstream component after the sources. So in this case, it would be the Join component. Use the paging features to see the inputs to the join processor.



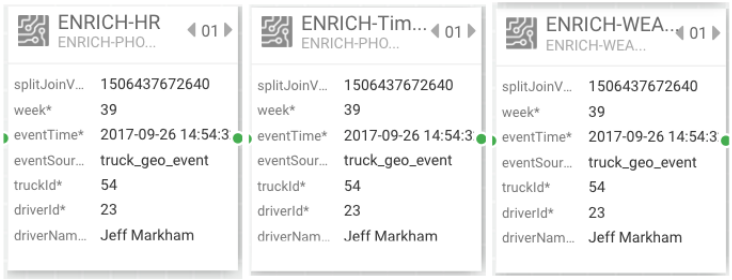
- Assertion 2 is to validate the Join of data between geo stream and speed stream. For this assertion, you would look at the downstream component after the Join. So in this case, it would be the EventType component. Note that you see speed and geo information.



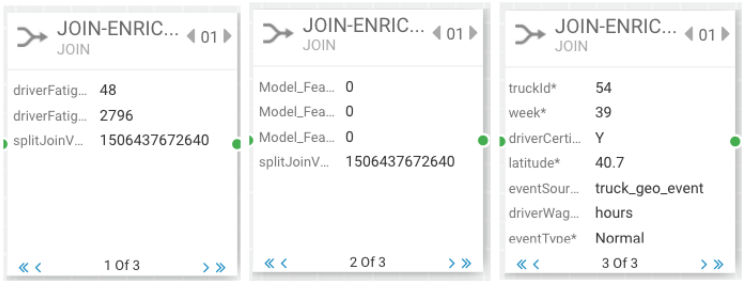
- Assertion 3 is to validate that the filter "EventType" detects that this is a "Non Violation Event". View the Split Component.



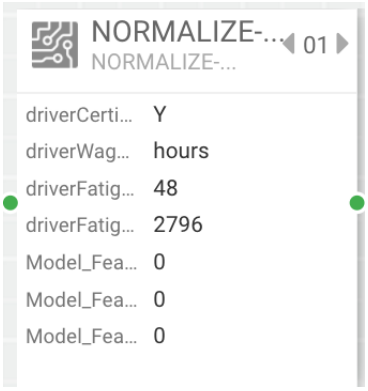
- Assertion 4 is to Validate test data for geo steam and speed stream that are non violations. View the JOIN-ENRICHMENT component



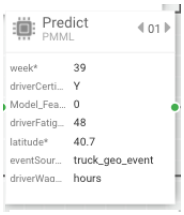
- Assertion 5 is to validate that the three enrichments are applied: weather enrichments, timesheet enrichment and HR enrichment. Use the paging features to page through the three enrichment outputs.



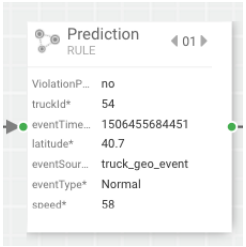
- Assertion 6 is to validate the three enrichment streams are joined into a single stream. View the NORMALIZE component.



- Assertion 7 is to validate that data after normalization for the model. View the Predict component.



- Assertion 8 is to validate that the output of the Prediction model is that no violation is predicted. View the Prediction component.



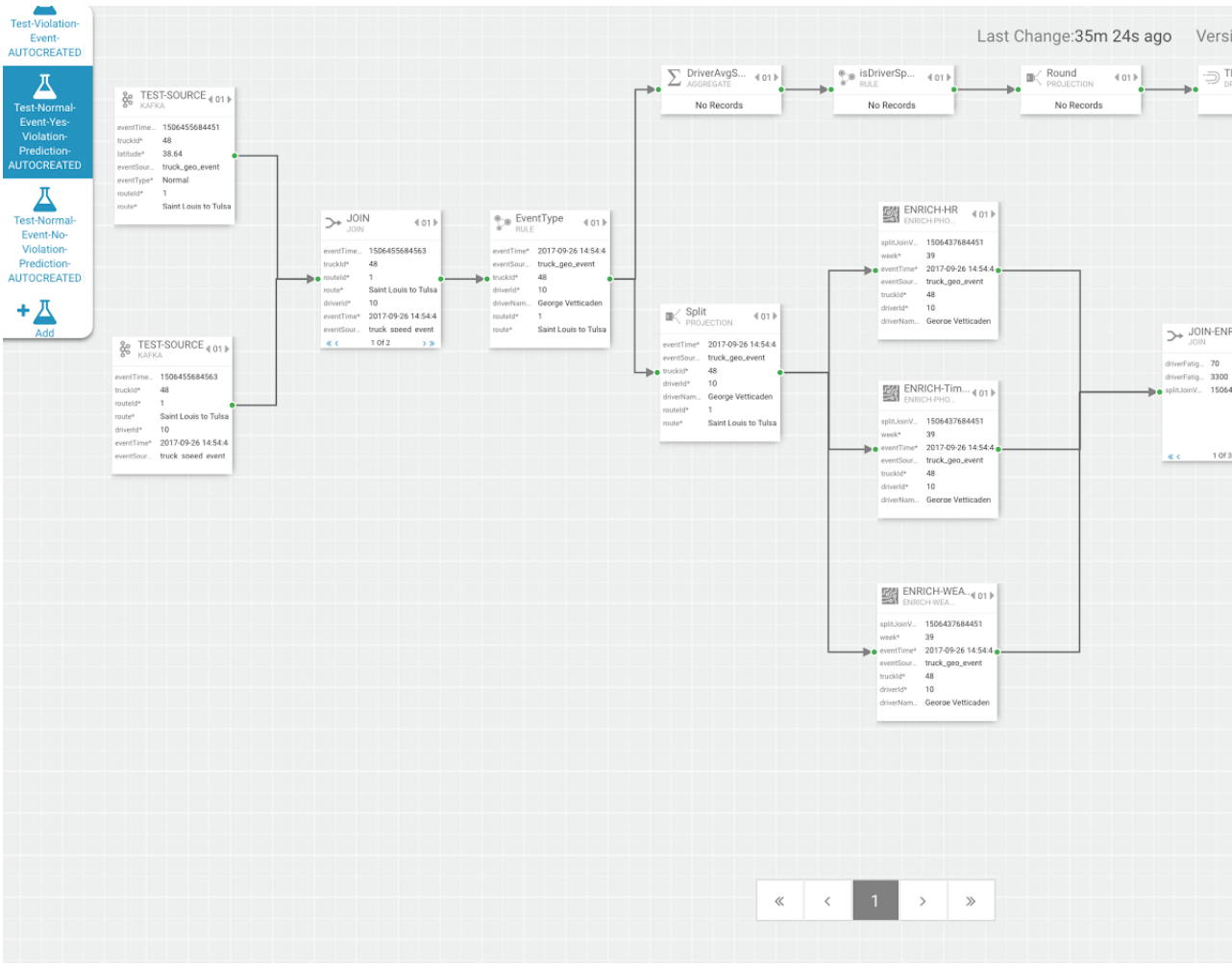
8.1.3. Test Case 2: Testing Normal Event with Yes Violation Prediction

In this test, we are validating all the same assertions as previous test but in this test case the violation prediction model should return true and be . Similar to above, Create a test named "Test-Normal-Event-Yes-Violation-Prediction", use the following test data for TruckGeoEvent and use the following test data for TruckSpeedEvent.

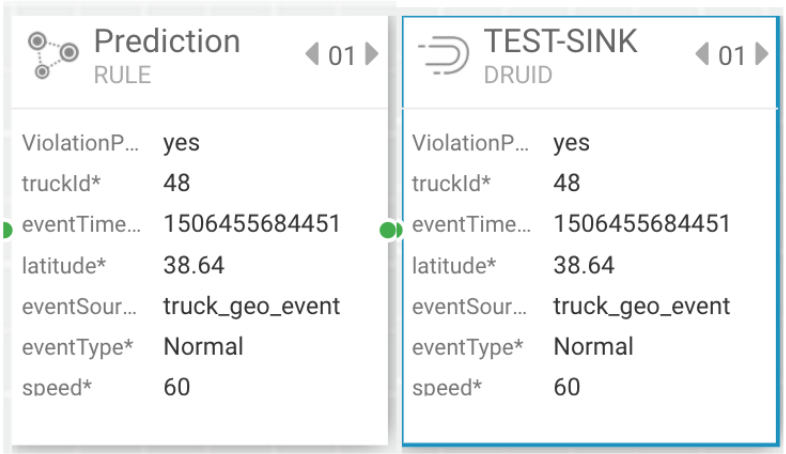
8.1.4. Analyzing Test Case 2 Results

1. Analyzing the Test Case Results.

   The output of the test case should be the following:



   To validate the PMML processor returns a violation prediction and sent to the sink, view the Prediction and Druid component.



8.1.5. Test Case 3: Testing Violation Event

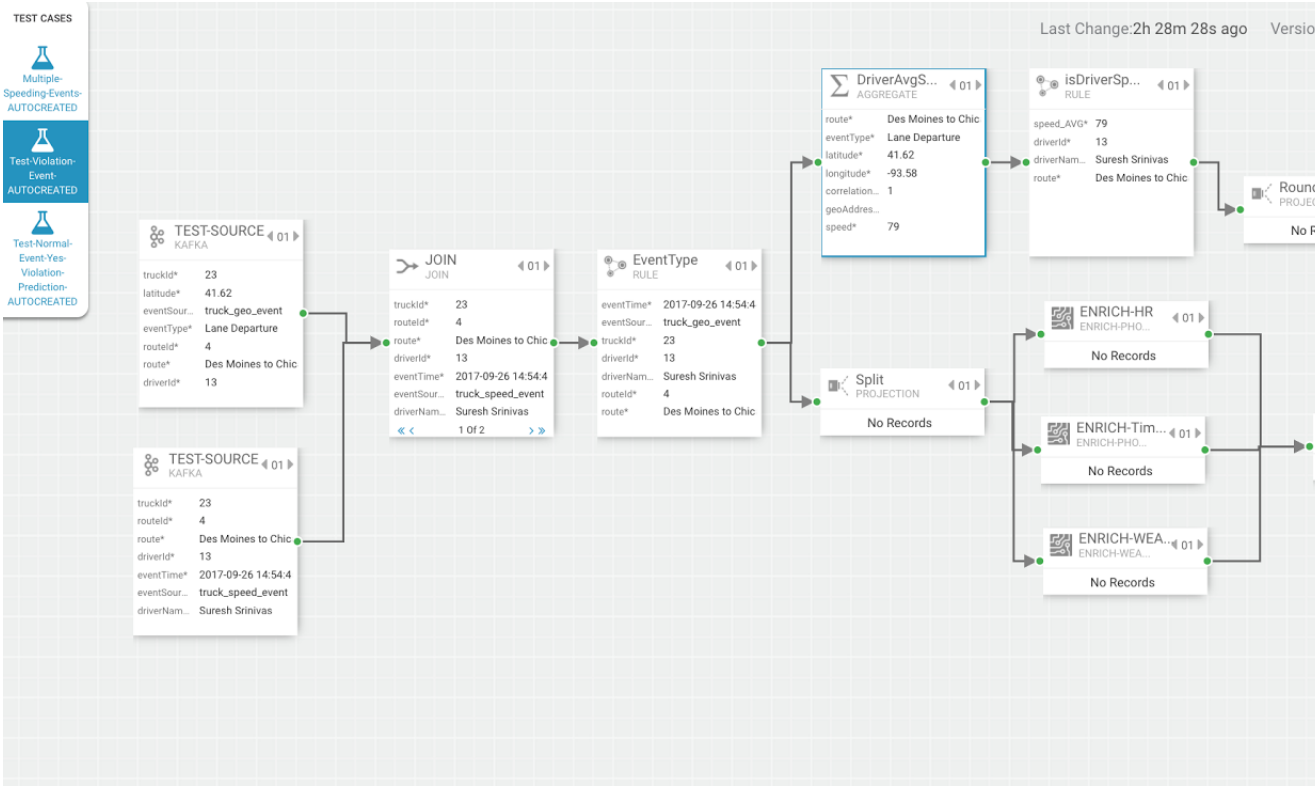
The Assertions of this test case are the following:

- Assertion 1: Validate test data for geo steam and speed stream that are “violation” events
- Assertion 2: Validate the Join of data between geo stream and speed stream
- Assertion 3: Validate that the filter “EventType” detects that this is a “Violation Event”
- Assertion 4: Validate that the inputs to the aggregate speed processor. There should only be 1 in the window
- Assertion 5: Validate the result of the DriverAvgSpeed aggregate process is average speed of 79 since there only 1 event
- Assertion 6: Validate the isDriverSpeeding rule recognized it was not speeding since the speed wasn’t greater than 80. The event should stop.

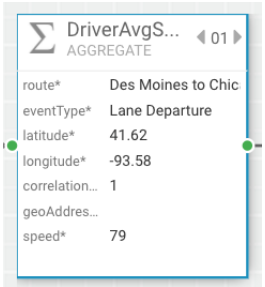
Create a test named “Test-Violation-Event”, use the following test data for TruckGeoEvent and use the following test data for TruckSpeedEvent.

8.1.6. Analyzing Test Case 3 Results

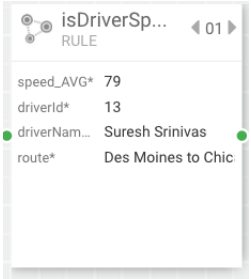
The output of the test case should look something like the following:



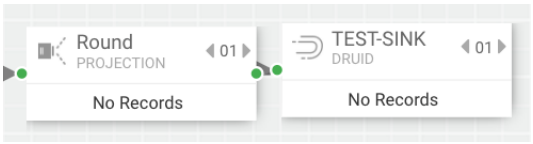
- Assertion 3 is to validate that the filter “EventType” detects that this is a “Violation Event” and Assertion 4 is to validate that the inputs to the aggregate speed processor should be 1 event within the window. View the DriverAvSpeed component to validate these assertions:



- Assertion 5 is to validate the result of the DriverAvgSpeed aggregate process is average speed of 79 since there only 1 event. View the isDriverSpeeding component:



- Assertion 6 is to validate the isDriverSpeeding rule recognized it was not speeding since the speed wasn't greater than 80. The event should stop. See the downstream components after isDriverSpeeding.



8.1.7. Test Case 4: Testing Multiple-Speeding-Events

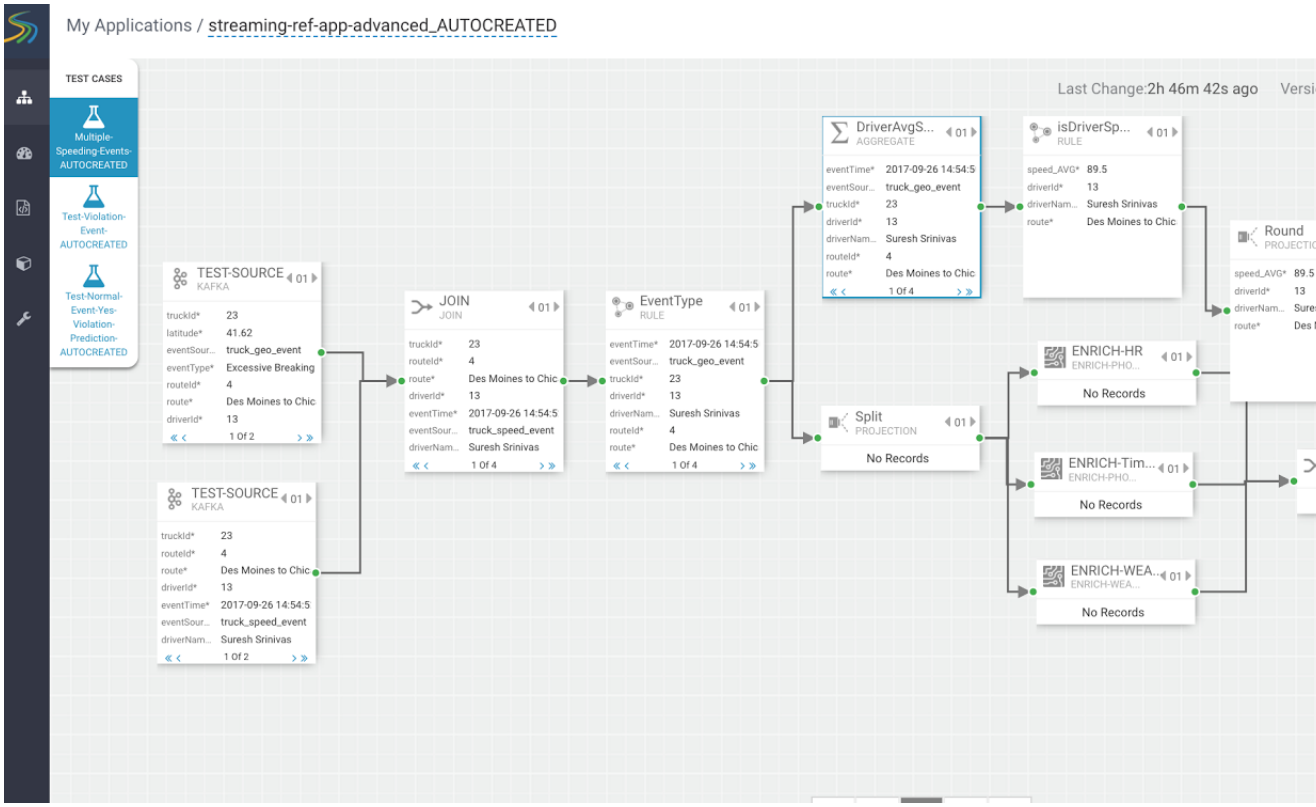
The Assertions of this test case are the following:

- Assertion 1: Validate that there are two geo events both of which are violations (Overspeed, Excessive Breaking) in source. Validate there are two speeding events both of which are speeding (96, 83)
- Assertion 2: Validate the Join of data between geo stream and speed streams
- Assertion 3: Validate that the filter "EventType" detects that this is a "Violation Event"
- Assertion 4: Validate the inputs of the window should be two events (geo/speed 1 with speed of 83, geo/speed 2 with speed of 96)
- Assertion 5: Validate the result of the DriverAvgSpeed aggregate processor should be one event that represents the average of 83 and 96...89.5
- Assertion 6: Validate the isDriverSpeeding rule recognizes it as speeding event (89.5) since it is greater than 80 and continue that event to custom round UDF
- Assertion 7: Validate the output of the round UDF event should change the speed from 89.5 to 90.0 and that is the final event that goes to the sink.

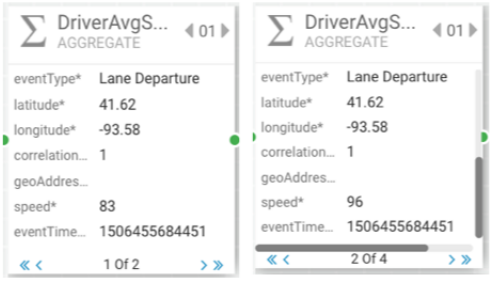
Create a test named "Test-Multiple-Speeding-Events", use the following test data for TruckGeoEvent and use the following test data for TruckSpeedEvent.

8.1.8. Analyzing Test Case 4 Results

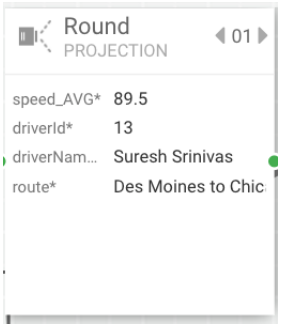
The output of the test case should look something like the following:



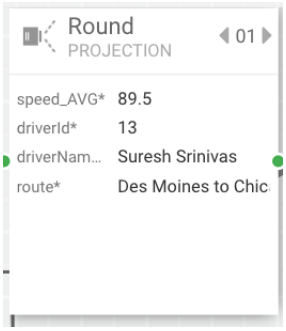
- Assertion 4 is to validate the inputs of the window should be two events (geo/speed 1 with speed of 83, geo/speed 2 with speed of 96). View the two events in the Join processor (use the paging feature to see the events)



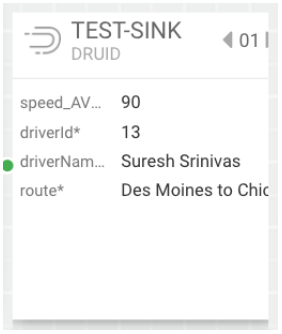
- Assertion 5 is to validate the result of the DriverAvgSpeed aggregate processor should be one event that represents the average of 83 and 96...89.5. View the Round Projection processor.



- Assertion 6 is to validate the isDriverSpeeding rule recognizes it as speeding event (89.5) since it is greater than 80 and continue that event to custom round UDF. View the Round Projection processor



- Assertion 7 is to validate the output of the round UDF event should change the speed from 89.5 to 90.0 and that is the final event that goes to the sink. View the Druid Test Sink component.



8.1.9. Running SAM Test Cases as Junit Tests in CI Pipelines

Using SAM's Test Mode provides a quick and effective way to test your applications locally visualizing the output within each component of the app without deploying to a cluster. Since all of SAM's capabilities is backed by REST apis, you can execute the these SAM Test Cases as part of your Junit Tests. This provides the power of using Junit assertions to validate the results of the test and incorporating them in automated tests as part of your continuous integration and delivery pipelines.

Examples of incorporating SAM Test Cases as part of unit tests can be found in the following artifacts:

- Trucking Ref App Git Hub Project
- TruckingRefAppAdvancedApp Junit Test Case

The Junit Test case above uses the SAM SDK project to setup a self contained Junit test that executes the SAM test cases and validates the result. The test case performs the following on setup of the this Junit Test Case:

- Create SAM Service Pool
- Create SAM Environment
- Import the Trucking Ref App

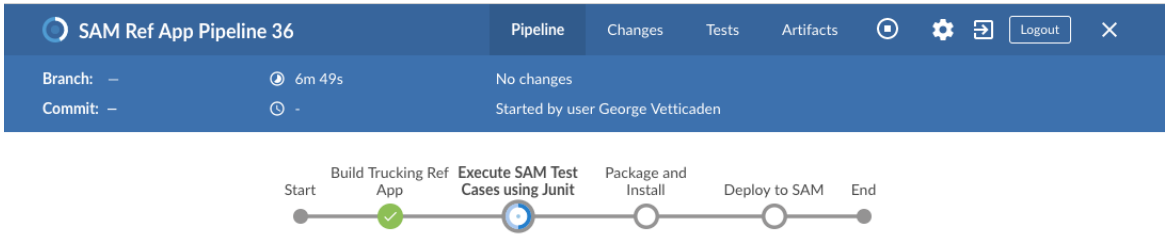
Then the following 4 test cases are executed:

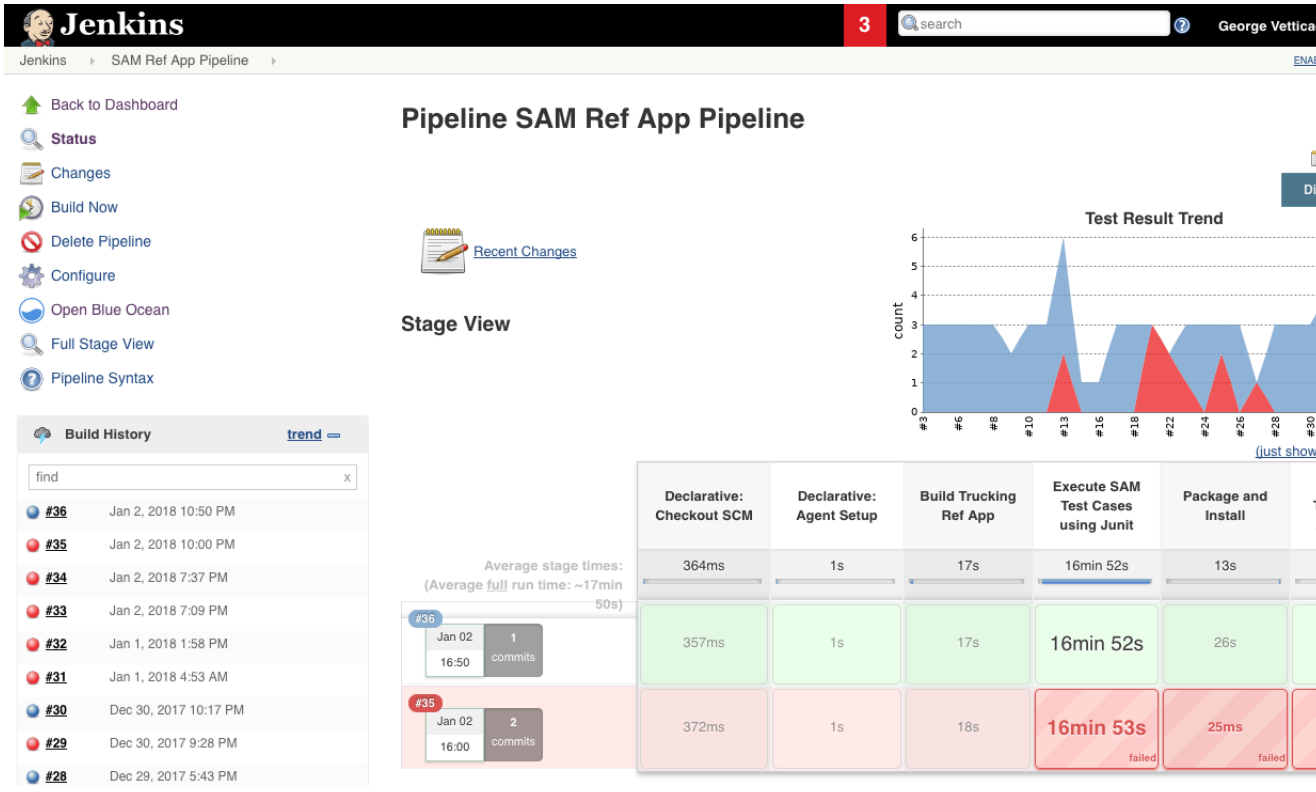
- testNormalEventNoViolationPrediction
- testNormalEventYesViolationPrediction
- testViolationTruckingEvents
- testMultipleSpeedingEvents

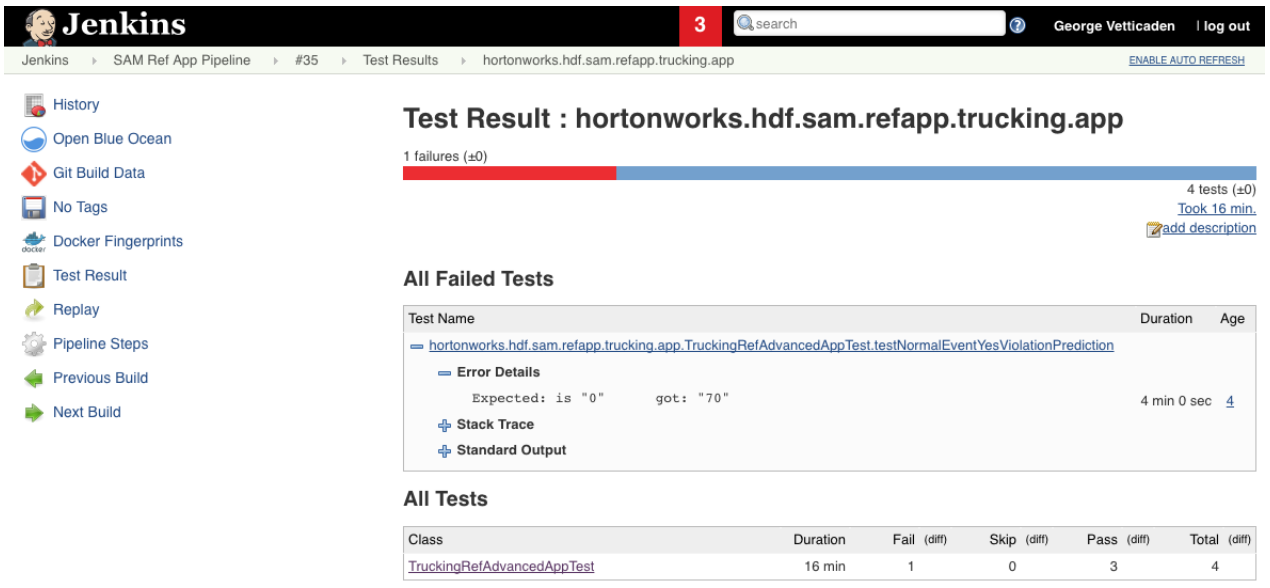
Each of these test cases will do the following:

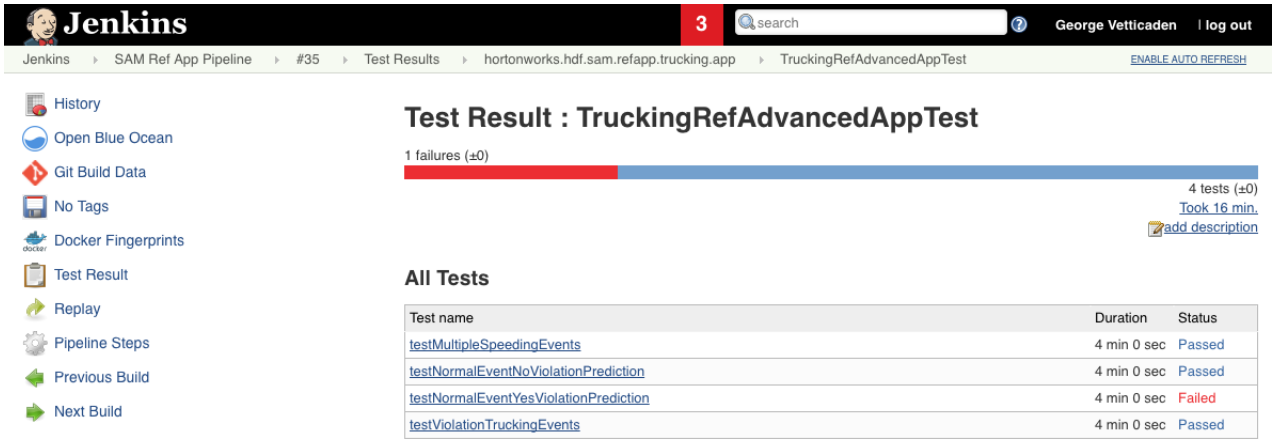
- Create the SAM Test Case
- Setup test data for each of the sources for each test case.
- Execute the SAM Test Case using SAM Test Mode and wait for test to complete.
- Download the results of the test case.
- Validate the results of the Test Case.

SAM Test Mode execution via Junit Tests allows you to integrate these tests as part of your continuous integration / delivery pipeline.




Jenkins
3
search
George Vettica
Jenkins › SAM Ref App Pipeline ›
Back to Dashboard
Status
Changes
Build Now
Delete Pipeline
Configure
Open Blue Ocean
Full Stage View
Pipeline Syntax
Build History
trend
find
#36 Jan 2, 2018 10:50 PM
#35 Jan 2, 2018 10:00 PM
#34 Jan 2, 2018 7:37 PM
#33 Jan 2, 2018 7:09 PM
#32 Jan 1, 2018 1:58 PM
#31 Jan 1, 2018 4:53 AM
#30 Dec 30, 2017 10:17 PM
#29 Dec 30, 2017 9:28 PM
#28 Dec 29, 2017 5:43 PM
Pipeline SAM Ref App Pipeline
Recent Changes
Stage View
Test Result Trend
count
(just show
Declarative: Checkout SCM
Declarative: Agent Setup
Build Trucking Ref App
Execute SAM Test Cases using Junit
Package and Install
Average stage times:
(Average full run time: ~17min 50s)
364ms
1s
17s
16min 52s
13s
#36
Jan 02 16:50
1 commits
357ms
1s
17s
16min 52s
26s
#35
Jan 02 16:00
2 commits
372ms
1s
18s
16min 53s
failed
25ms
failed


Jenkins
3
search
George Vetticaden
log out
Jenkins › SAM Ref App Pipeline › #35 › Test Results › hortonworks.hdf.sam.refapp.trucking.app
ENABLE AUTO REFRESH
History
Open Blue Ocean
Git Build Data
No Tags
Docker Fingerprints
Test Result
Replay
Pipeline Steps
Previous Build
Next Build
Test Result : hortonworks.hdf.sam.refapp.trucking.app
1 failures (±0)
4 tests (±0)
Took 16 min.
add description
All Failed Tests
Test Name
Duration
Age
hortonworks.hdf.sam.refapp.trucking.app.TruckingRefAdvancedAppTest.testNormalEventYesViolationPrediction
Error Details
Expected: is "0" got: "70"
4 min 0 sec
4
Stack Trace
Standard Output
All Tests
Class
Duration
Fail (diff)
Skip (diff)
Pass (diff)
Total (diff)
TruckingRefAdvancedAppTest
16 min
1
0
3
4


Jenkins
3
search
George Vetticaden
log out
Jenkins
SAM Ref App Pipeline
#35
Test Results
hortonworks.hdf.sam.refapp.trucking.app
TruckingRefAdvancedAppTest
ENABLE AUTO REFRESH
History
Open Blue Ocean
Git Build Data
No Tags
Docker Fingerprints
Test Result
Replay
Pipeline Steps
Previous Build
Next Build
Test Result : TruckingRefAdvancedAppTest
1 failures (±0)
4 tests (±0)
Took 16 min.
add description
All Tests
Test name
Duration
Status
testMultipleSpeedingEvents
4 min 0 sec
Passed
testNormalEventNoViolationPrediction
4 min 0 sec
Passed
testNormalEventYesViolationPrediction
4 min 0 sec
Failed
testViolationTruckingEvents
4 min 0 sec
Passed

9. Creating Custom Sources and Sinks

Throughout the getting started doc with the trucking reference application, we have showcased the powerful extensibility features of SAM including:

- Uploading custom UDFs
- Uploading custom processors

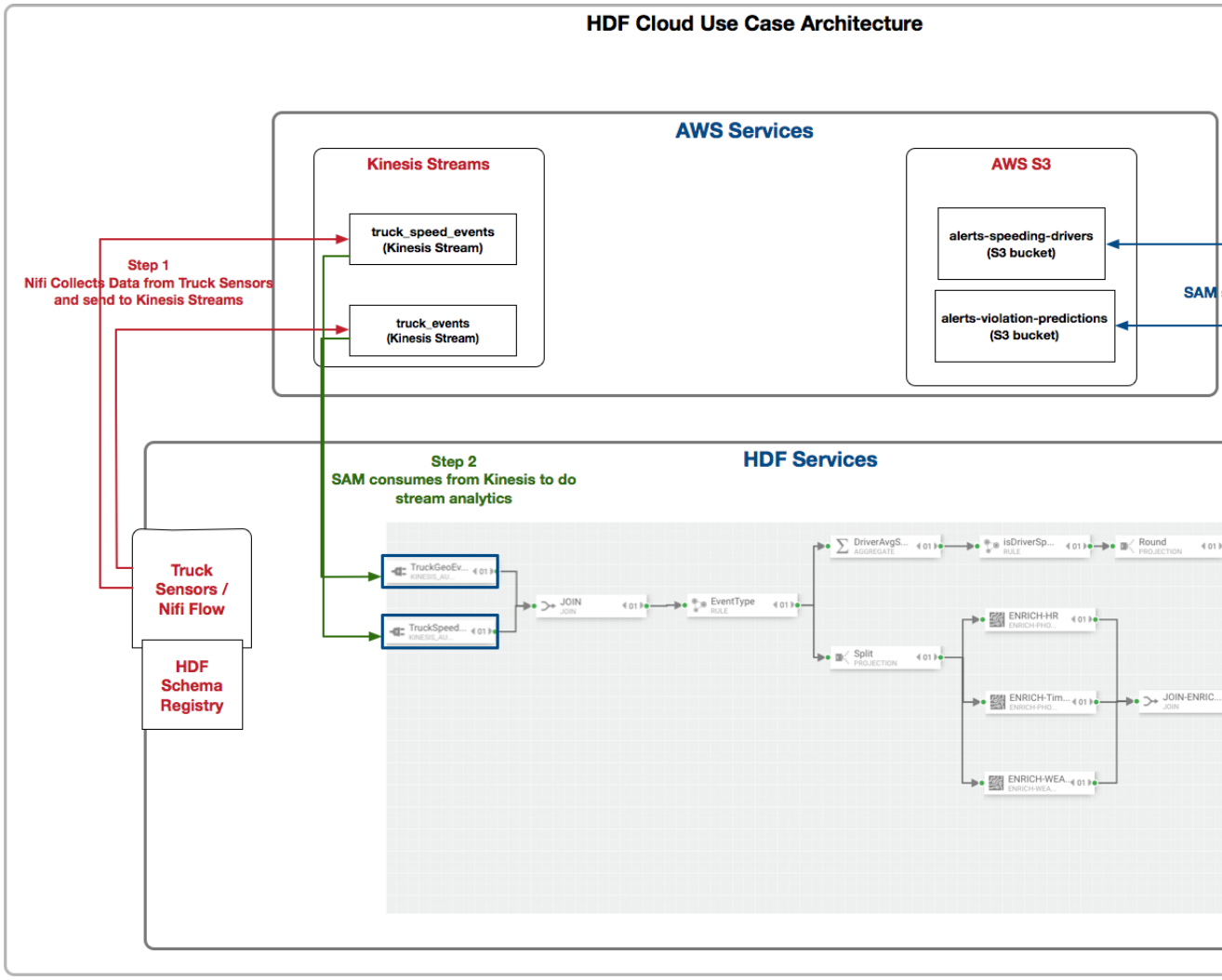
In this section, we walk through how to register custom sources and sinks in SAM integrated with Schema Registry.

9.1. Cloud Use Case: Integration with AWS Kinesis and S3

To showcase registering custom sources and sink, lets modify our Trucking Ref App Use Case Requirements with the following:

- The Trucking company wants to deploy the Trucking Application on AWS
- The Trucking company wants to streams the sensor data into AWs Kinesis instead of Apache Kafka.
- The trucking company wants to use SAM for streaming analytics.
- The insights generated by SAM should be stored into AWS S3 instead of Druid.

The below diagram illustrates this Cloud architecture:



9.2. Registering a Custom Source in SAM for AWS Kinesis

To register any custom source in SAM, there are three artifacts you need:

1. Artifact 1: Code for the custom source using the underlying streaming engine. Since SAM today supports Storm as the Streaming engine, you can refer to the following artifacts for the custom source:

   - Git Project for Storm Kinesis
   - AWS Storm Kinesis Spout

2. Artifact 2: Code for mapping the SAM configs to the custom source/spout. Refer to the following artifacts for this mapping code:

   - Git Project for SAM Storm Kinesis Mapping

- SAM Kinesis Flux Mapping Class

3. Artifact 3: Flux mapping file to map the SAM config to the Kinesis Spout. Refer to the following artifacts

   - SAM Kinesis Flux Mapping Config

More Details on implementing a custom source and registering with SAM can be found here: https://github.com/hortonworks/streamline/tree/master/examples/sources

To register the custom Kinesis Source in SAM using the above three artifacts, perform the following steps:

1. Download the Sam-Custom-Extensions.zip to the host where SAM is installed (if you haven't done it in a past step)

2. Unzip the contents. We will call the unzipped folder $SAM_EXTENSIONS

3. Switch to user streamline:

```
sudo su streamline
```

4. Install Artifact 1 (the custom source code) on host's local maven repo

```
cd $SAM_EXTENSIONS/custom-source/kinesis/
mvn install:install-file -Dfile=storm-kinesis-1.1.0.5.jar \
-DgroupId=org.apache.storm \
-DartifactId=storm-kinesis \
-Dversion=1.1.0.5 \
-Dpackaging=jar
```

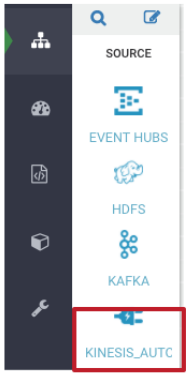
5. Register the custom source via SAM REST call. Replace SAM_HOST and SAM_PORT.

```
curl -sS -X POST -i -F \
topologyComponentBundle=@config/kinesis-source-topology-component.json -F \
bundleJar=@sam-custom-source-kinesis.jar \
http://SAM_HOST:SAM_PORT/api/v1/catalog/streams/componentbundles/SOURCE
```

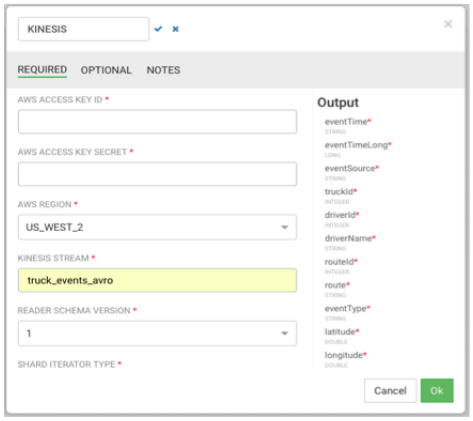
6. If the registration was successful, you should see a message like the following by the REST response:

```
HTTP/1.1 201 Created
Date: Wed, 03 Jan 2018 20:26:22 GMT
Content-Type: application/json
Content-Length: 4569
```

7. On the SAM Application Canvas Palette, you should now see KINESIS source.



8. Dragging the kinesis source onto the canvas and double clicking it, you should see the following kinesis dialog. The dialog properties comes from the topologyComponentBundle flux config you used to register the custom source.



9.3. Registering a Custom Sink in SAM for AWS S3

To register any custom sink in SAM, there are three artifacts you need:

1. Artifact 1: Code for the custom sink using the underlying streaming engine. Since SAM today supports Storm as the Streaming engine, you can refer to the following artifacts for the custom sink:
   - Git Project for Storm S3
   - Storm S3 Sink
2. Artifact 2: Code for mapping the SAM configs to the custom/spout. Refer to the following artifacts for this mapping code:
   - Git Project for SAM Storm S3 Mapping
   - SAM S3 Flux Mapping Class
3. Artifact 3: Flux mapping file to map the SAM config to the S3 Sink. Refer to the following artifacts

- SAM S3 Flux Mapping Config

To register the custom S3 Sink in SAM using the above three artifacts, perform the following steps:

1. Download the Sam-Custom-Extensions.zip to the host where SAM is installed (if you haven't done it in a past step).

2. Unzip the contents. We will call the unzipped folder $SAM_EXTENSIONS.

3. Switch to user streamline.

```
sudo su streamline
```

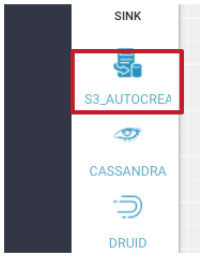
4. Install Artifact 1 (the custom sink/bolt code) on host's local maven repo.

```
cd $SAM_EXTENSIONS/custom-sink/s3

mvn install:install-file \
-Dfile=storm-s3-0.0.1-SNAPSHOT.jar \
-DgroupId=hortonworks.storm.aws \
-DartifactId=storm-s3 \
-Dversion=0.0.1-SNAPSHOT \
-Dpackaging=jar
```

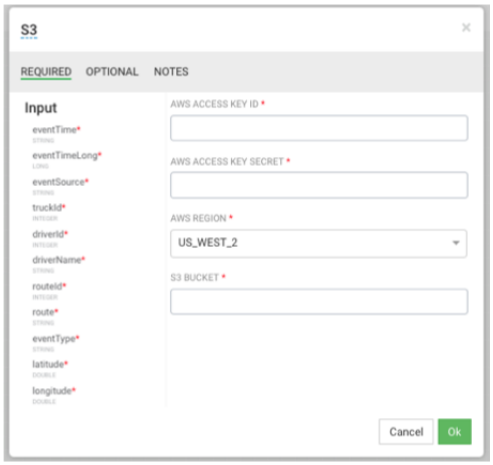
5. Register the custom sink via SAM REST call. Replace SAM_HOST and SAM_PORT.

```
curl -sS -X POST -i -F \
topologyComponentBundle=@config/s3-sink-topology-component.json -F \
bundleJar=@sam-custom-sink-s3.jar \
http://SAM_HOST:SAM_PORT/api/v1/catalog/streams/componentbundles/SINK
```

6. On the SAP App Canvas Palette, you should now see S3 sink.



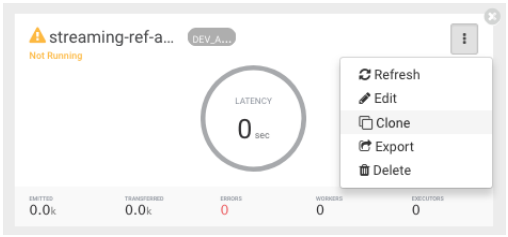
7. Dragging the S3 sink onto the canvas and double clicking it, you should see the following s3 dialog. The dialog properties comes from the topologyComponentBundle flux config you used to register the custom sink.



9.4. Implementing the SAM App with Kinesis Source and S3 Sink

Now that we have registered teh custom Kinesis and S3 sources and sink, we can now build the streaming application in SAM to implement the cloud use case requirements.

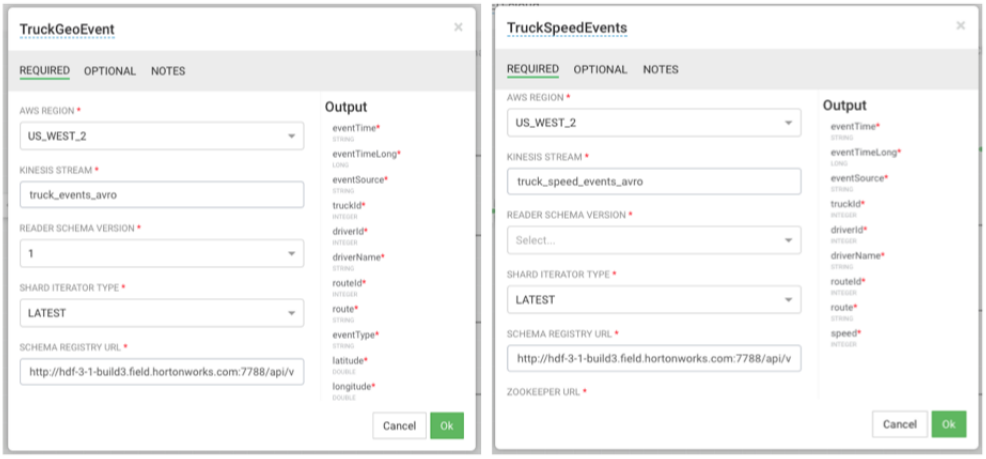
1. Clone the trucking ref app



2. Rename the clone app to streaming-ref-app-advanced-cloud

3. Delete the Kafka sources and druid sinks from the SAM App

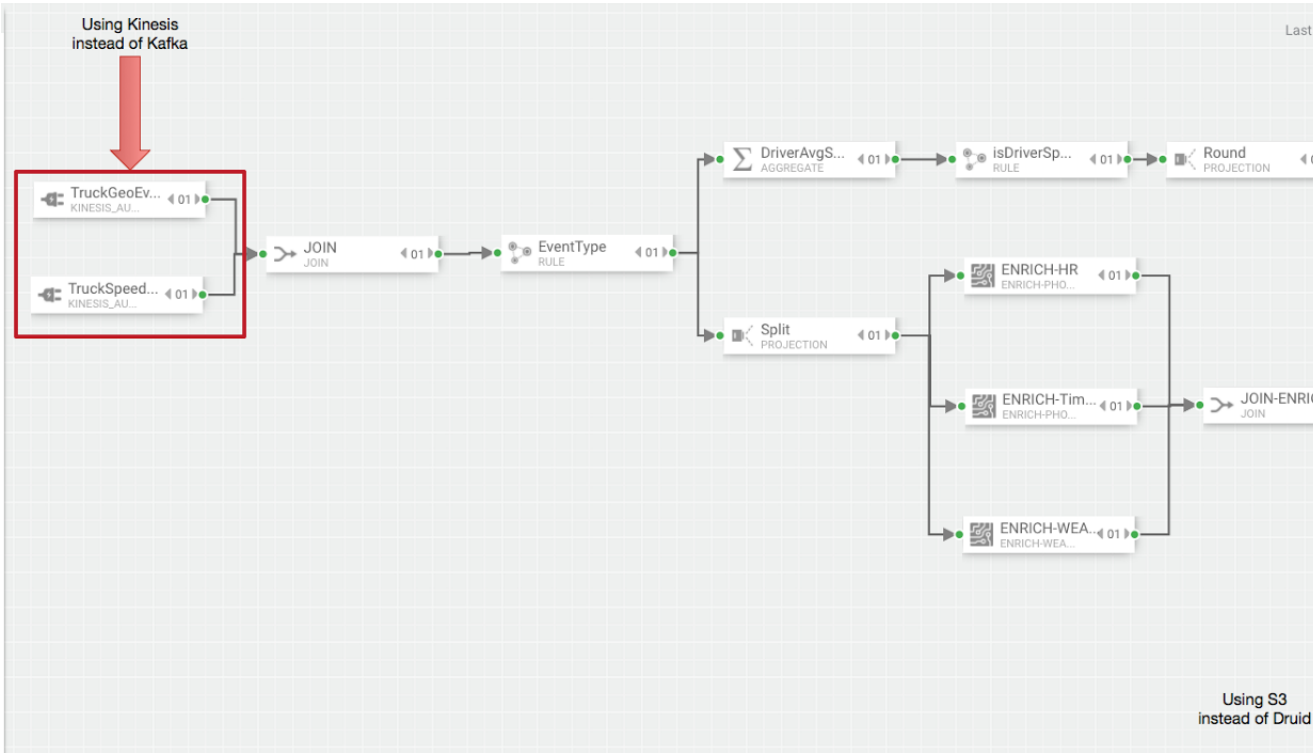
4. Add Kinesis sources for the deleted the Kafka Topics. Make sure to create the create the kinesis streams in AWS with the same names as the schemas you defined SAM's SR. You you will need to reevaluate the config for other components that are marked as yellow.



5. Add S3 sink for the deleted druid sinks. Make sure to create the S3 buckets in AWS. If you can't connect to the S3 from the Round projection, try deleting the Round projection, adding it back in and then connecting it to the S3.

6. Remove any HDFS or HBase Sinks that you have in the app.

The SAM App should look like the following:



10. 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

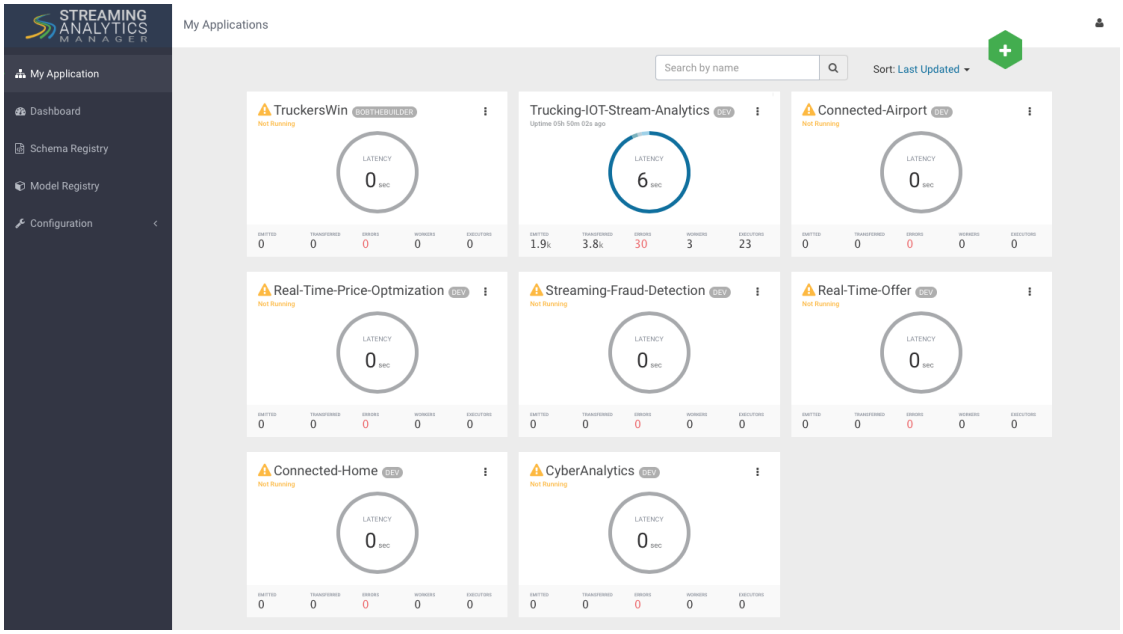
10.1. My Applications View

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.

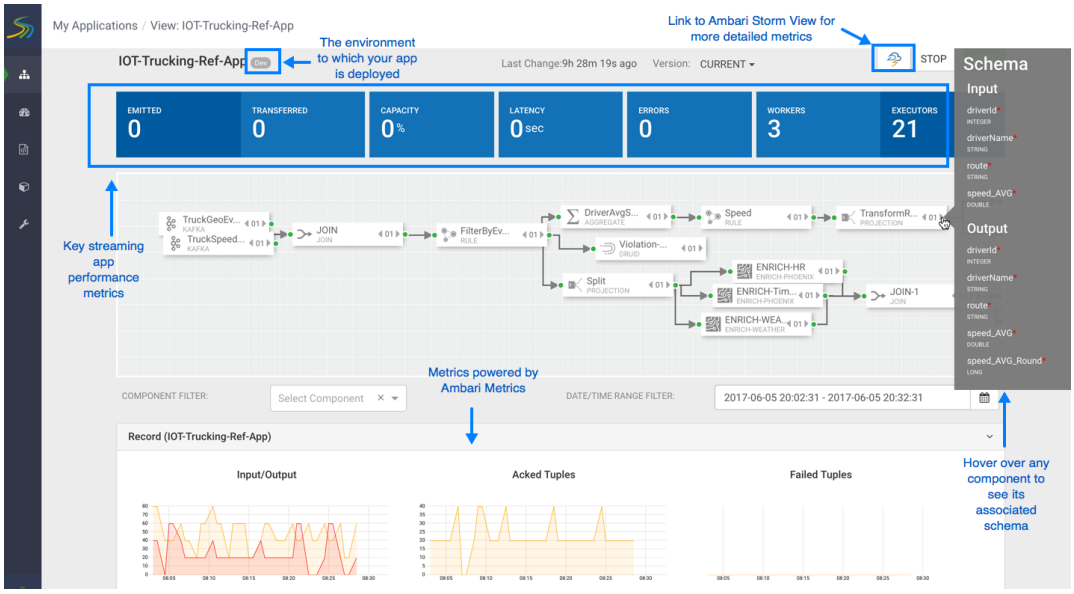
Each stream application is represented by an application tile. Hovering over the application tile displays status, metrics, and actions you can perform on the stream application.



10.2. Application Performance Monitoring

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

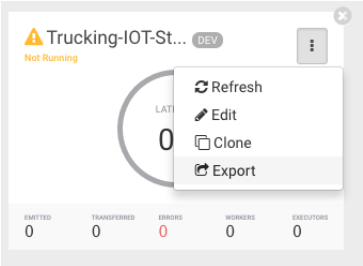
The following diagram describes elements of the APM view.



10.3. Exporting and Importing Stream Applications

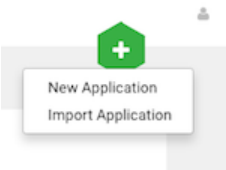
Service pool and environment abstractions combined with import and export capabilities allow you to move a stream application from one environment to another.

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

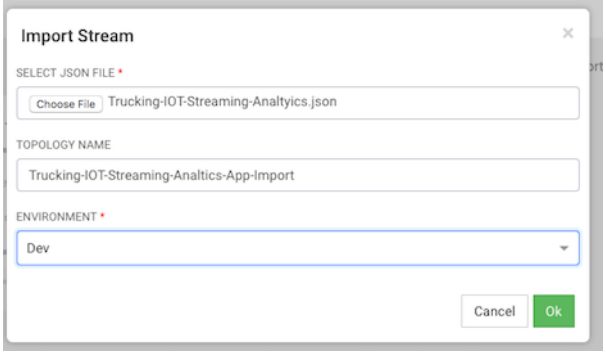


To import a stream application that was exported in JSON format:

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.



10.4. Troubleshooting and Debugging a Stream Application

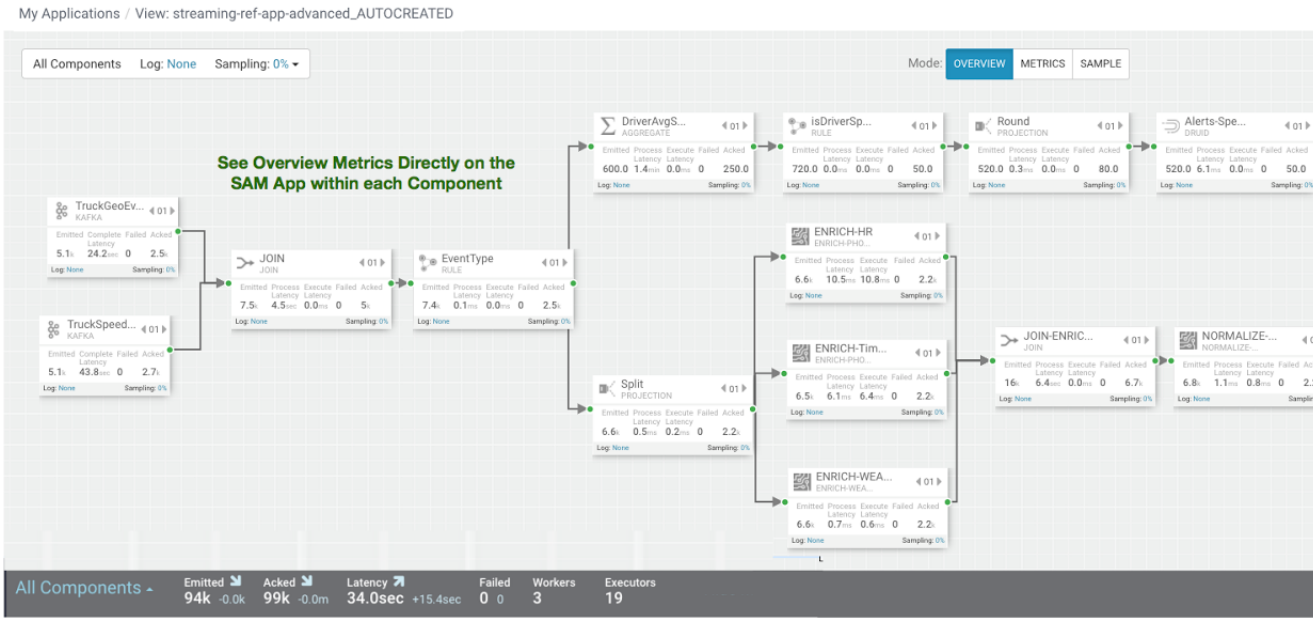
Once we have deployed the streaming app, common actions performed by users such as DevOps, Developers, and Operations teams are the following:

- Monitoring the Application and troubleshooting and identifying performance issues
- Troubleshooting an application through Log Search
- Troubleshooting an application through Sampling

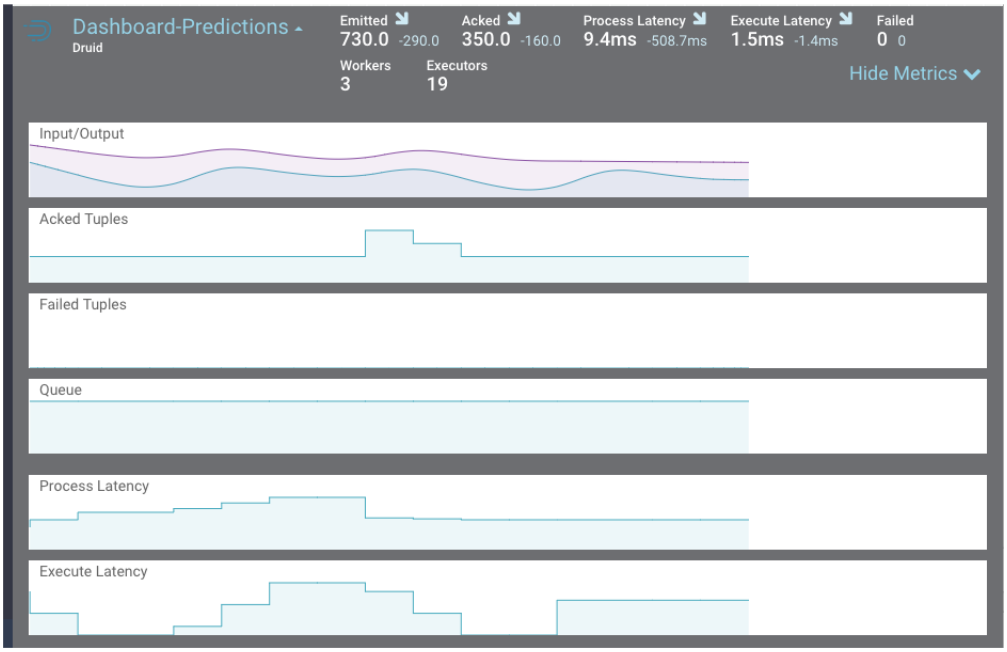
SAM makes performing these tasks easier by using the same visual approach as users have when developing the application. We will walk through these common use cases in the below sections.

10.4.1. Monitoring SAM Apps and Identifying Performance Issues

After deploying SAM and running the test generator for about 30 mins, your Storm Operation Mode of the app renders important metrics within each component on the canvas like below.



You can click on **Show Metrics** to get more details on the metrics and drill down on individual metrics. Note the detailed level metrics for **All Components**, **TruckGeoEvent Kafka** source, and **Dashboard-Predictions** Druid Sink.



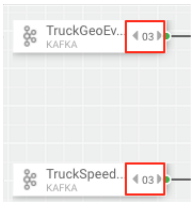
Key metrics include the following:

Metric Name	Description
Execute Latency	The average time it takes an event to be processed by a given component
Process Latency	The average time it takes an event to be acked. Bolts that join, aggregate or batch may not Ack a tuple until a number of other Tuples have been received
Complete Latency	How much time an event from source takes to be fully processed and acked by the topology. This metrics is only available for sources (e.g.: Kafka Source)
Emitted	The number of events emitted for the given time period. For example, for a Kafka Source, it is the number of events consumed for the given time period
Acked	The number of events acked for the given time period. For example, for a Kafka Source, it is the number of events consumed and then acked.

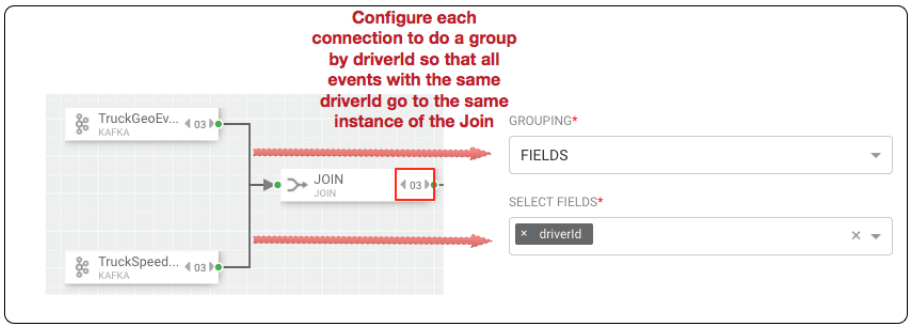
10.4.1.1. Identifying Throughput Bottlenecks

Looking through the metrics the Source and Sink metrics, we want to increase the throughput such that we emit/consume more events from the Kafka Topic and send more events to Druid sink over time. We make some changes to the app to increase throughput.

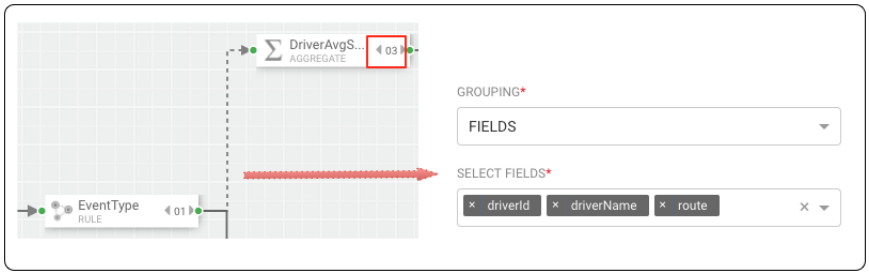
Increase the parallelism of TruckGeoEvent (kafka topic: truck_events_avro) and TruckSpeedEvent (kafka topic: truck_speed_events_avro) from 1 to 3. Note that each of these kafka topics have three partitions.



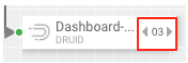
Increase the parallelism of the Join from 1 to 3. Since the join is grouped by driverId, we can configure the connection to use fields grouping to send all events with driverId to the same instance of the Join.



Increase the parallelism of the DriverAvgSpeed aggregate window from 1 to 3. Since the window groups by driverId,driverName and route, we can configure the connection to use fields grouping to send all events with those field values to the same instance of the window.

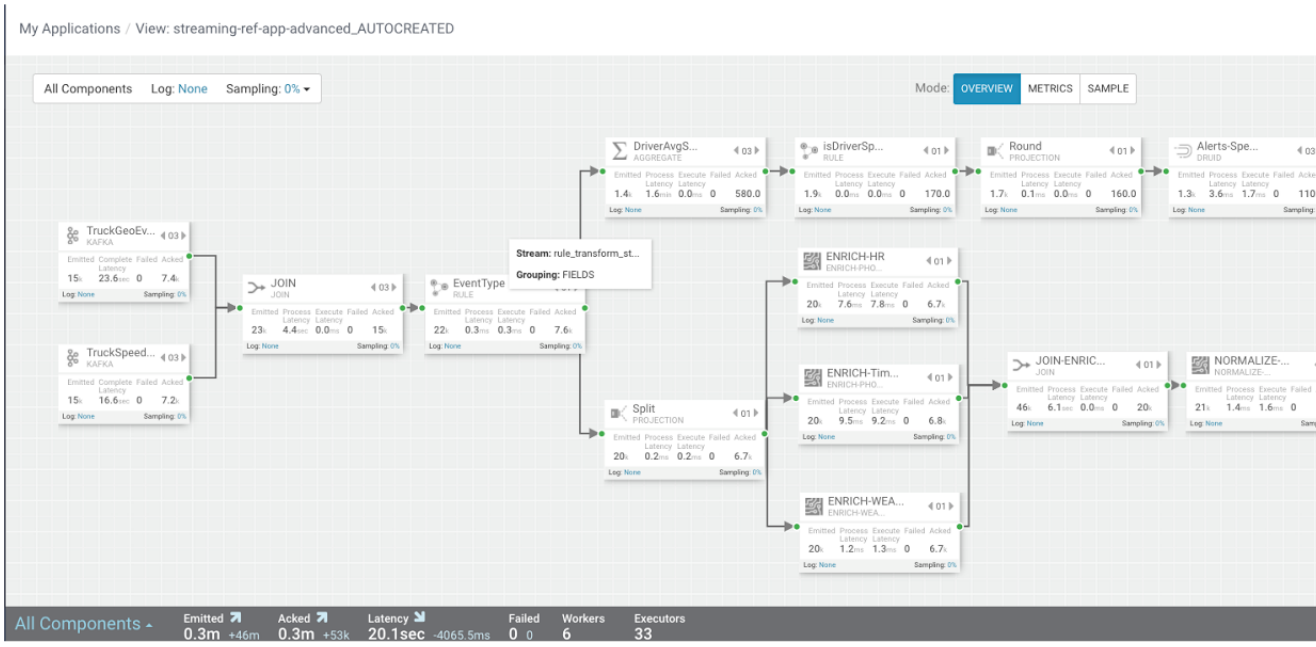


Increase the parallelism of the Dashboard-Predictions Druid sink from 1 to 3 so we can have multiple JVM instances of Druid writing to the cube.



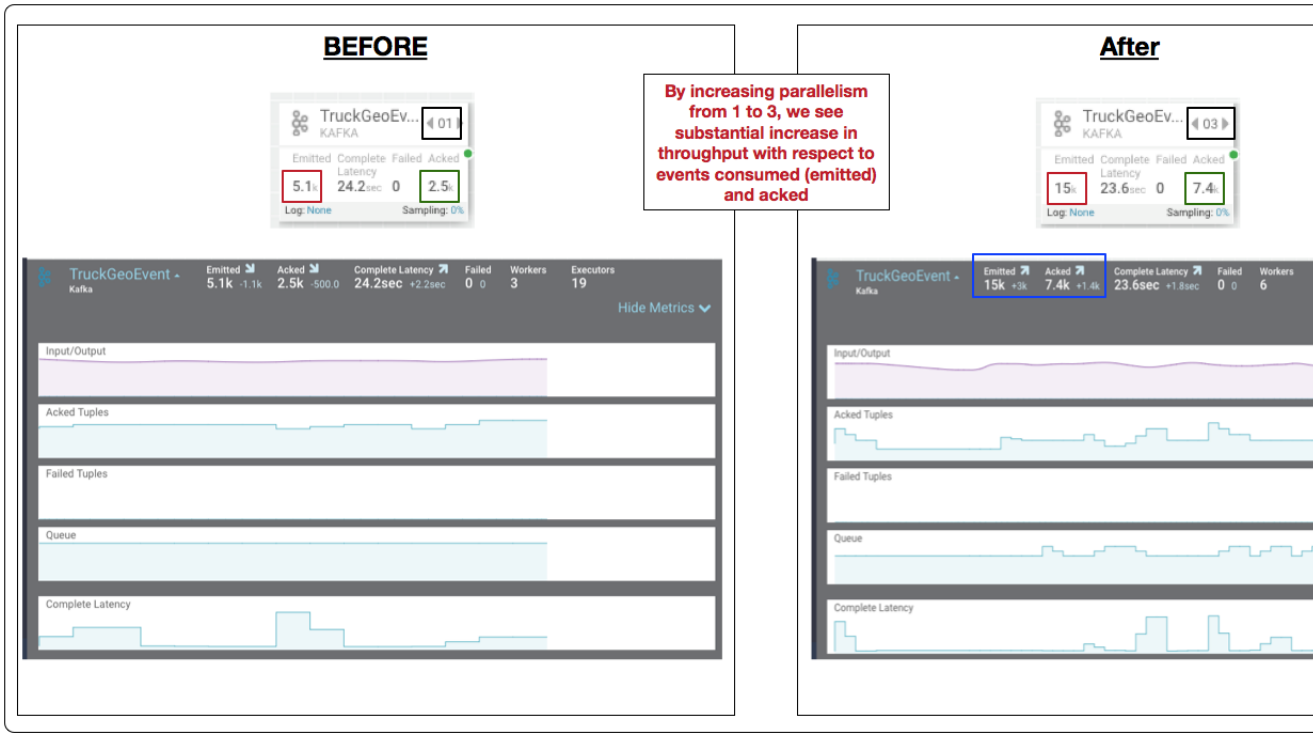
After making these changes, we re-deploy the app using SAM and run the data generator for about 15 minutes and view seeing the following metrics.

SAM's overview and detailed metrics makes it very easy to verify if the performance changes we made had the desired effect.

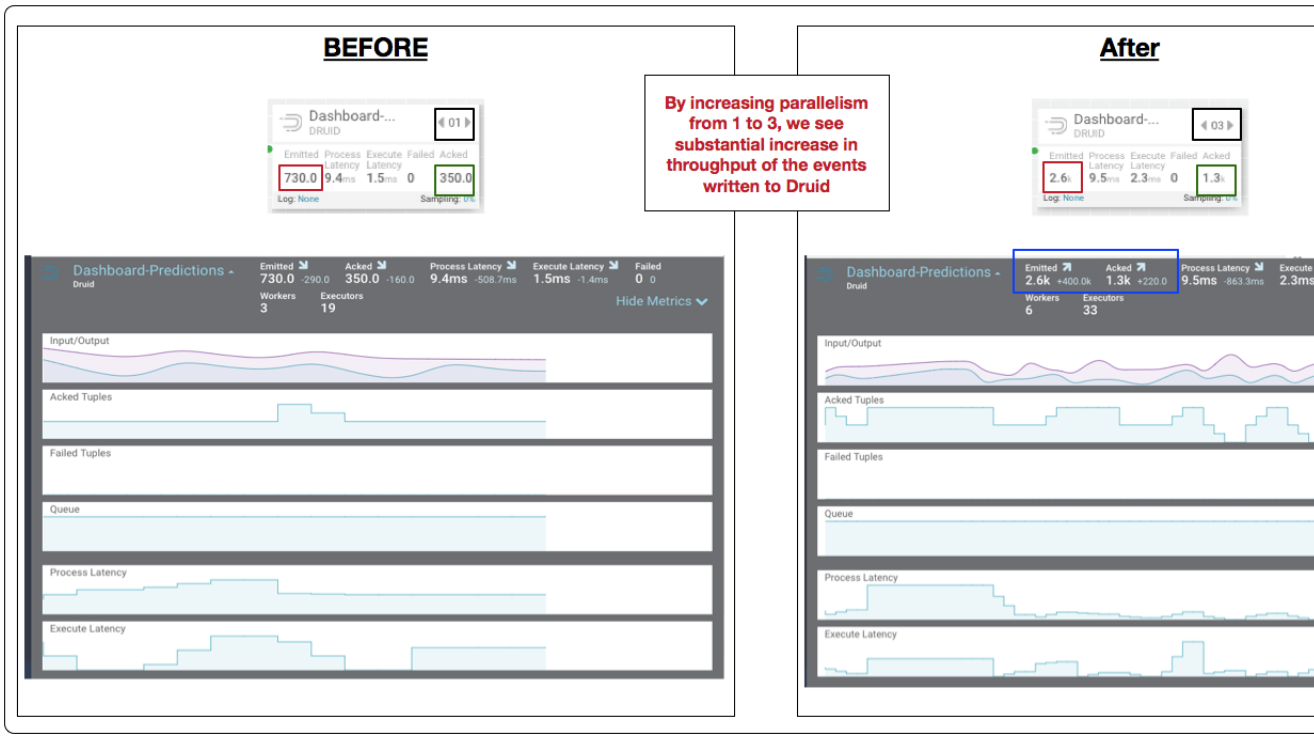


10.4.1.2. Throughput Improvements for the Kafka Source

The below is the before and after metrics for the TruckGeoEvent Kafka Sink:

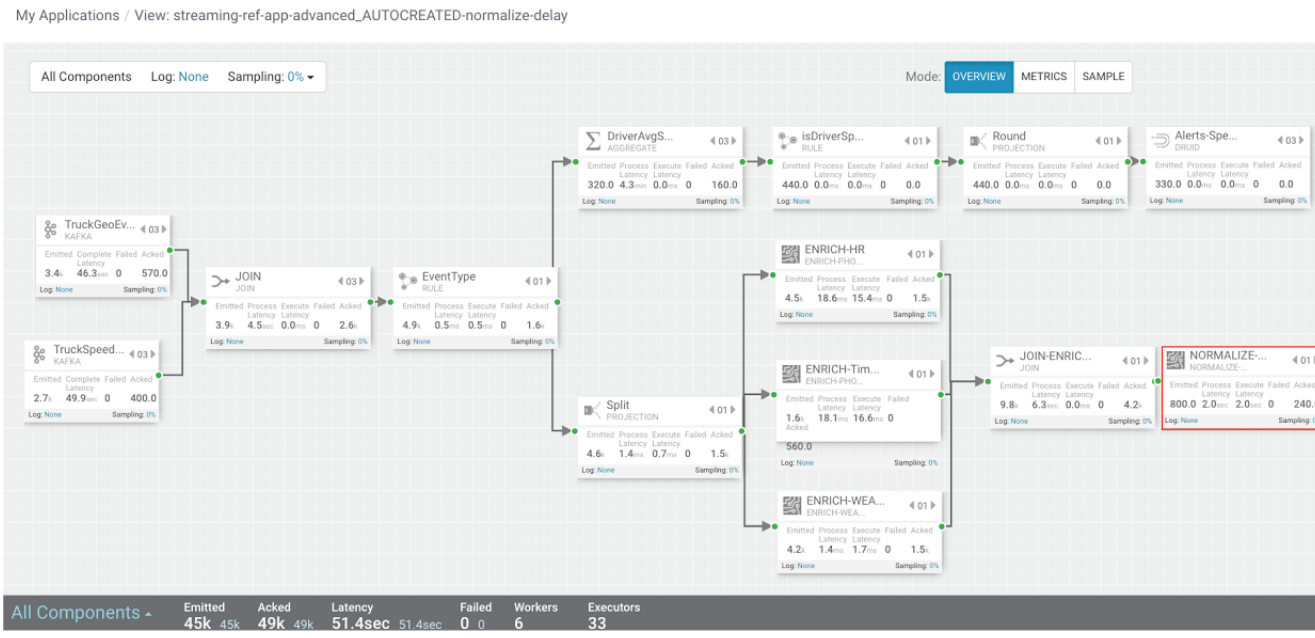


The below is the before and after metrics for the Dashboard-Predictions Druid Sink:

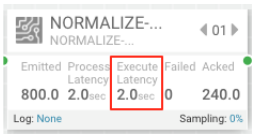


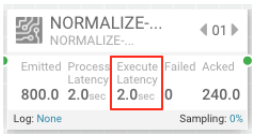
10.4.2. Identifying Processor Performance Bottlenecks

In this scenario, we identify a custom processor that has high latency. After running the data simulator for 30 mins, we view the Overview Metrics of the topology.

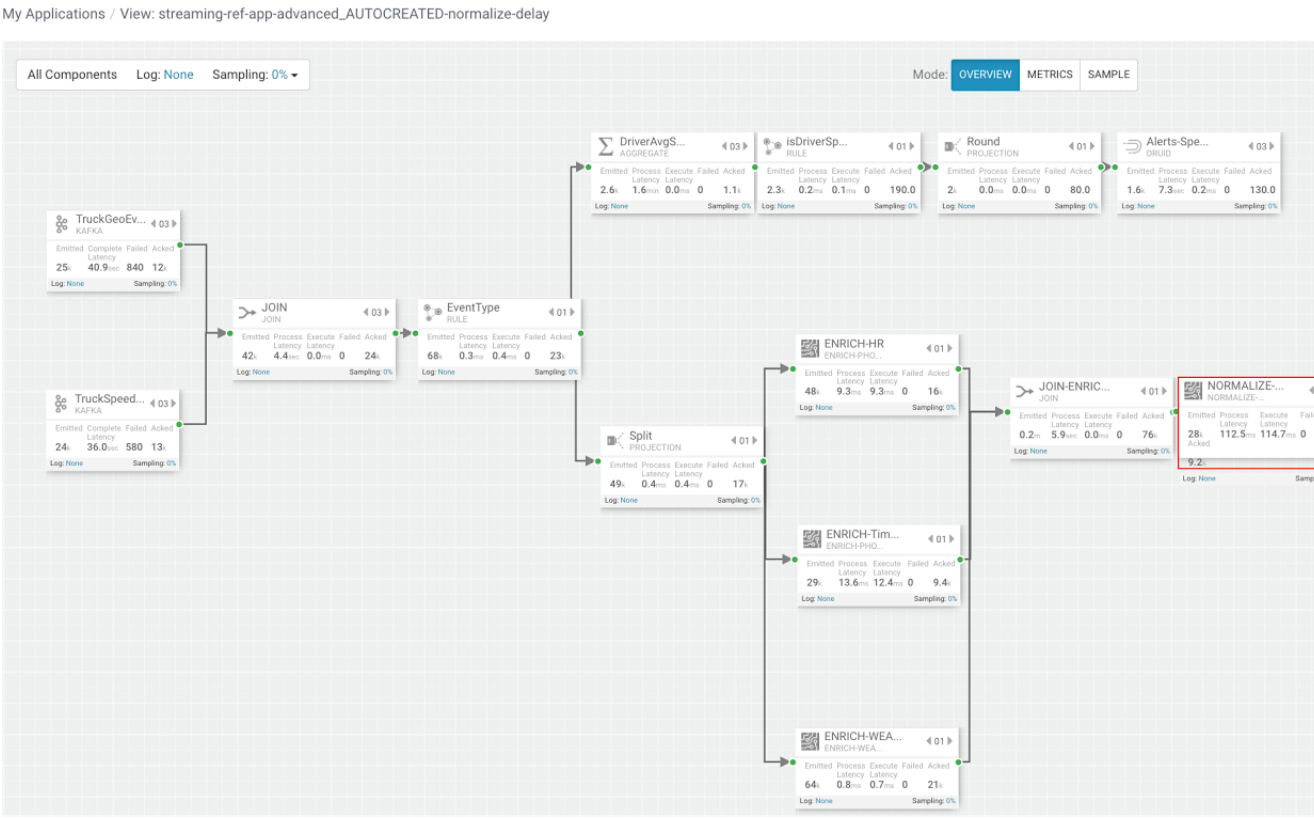


Scanning over the metrics, we see that the NORMALIZE-MODEL-FEATURES custom processor has high execute latency of 2 seconds. This means that over the last 30 minutes the average time an event spends in this component is 2 seconds.





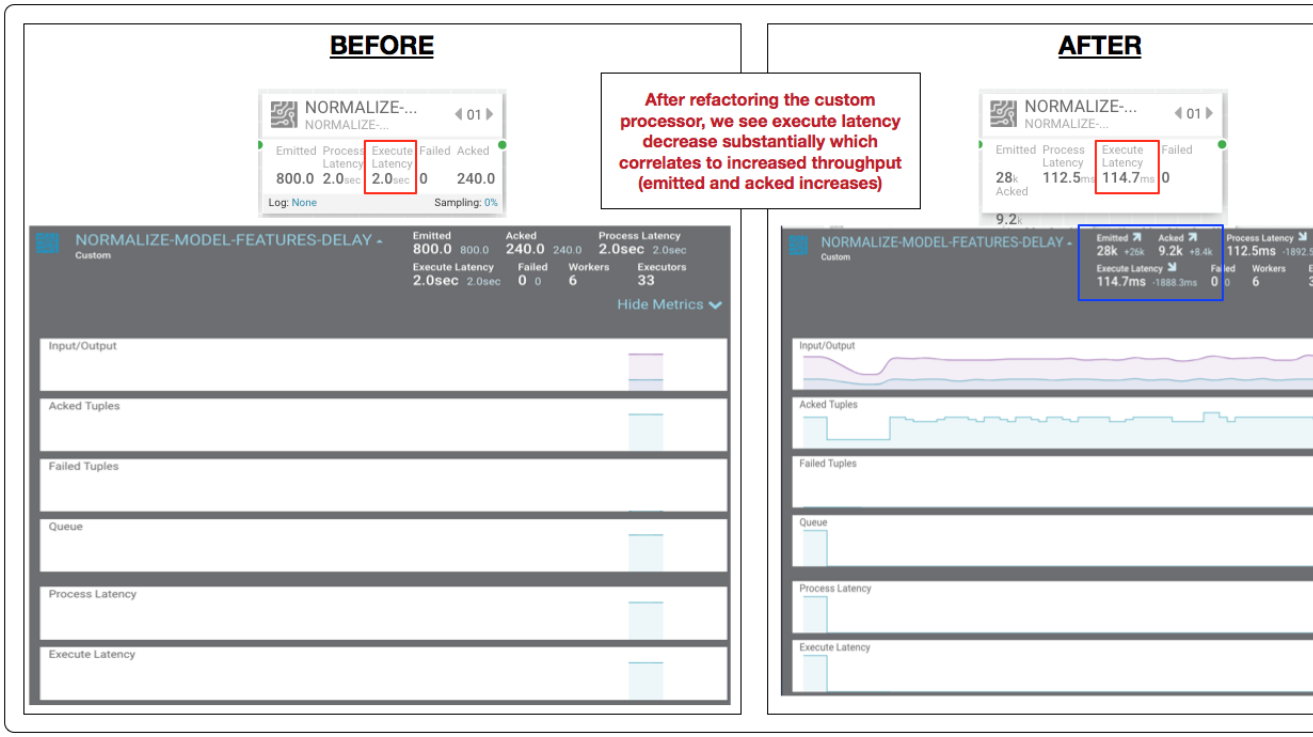
After making changes to the custom processor to address the latench, we re-deploy the app via SAM and run the data generator for about 15 minutes and view seeing the following metrics.



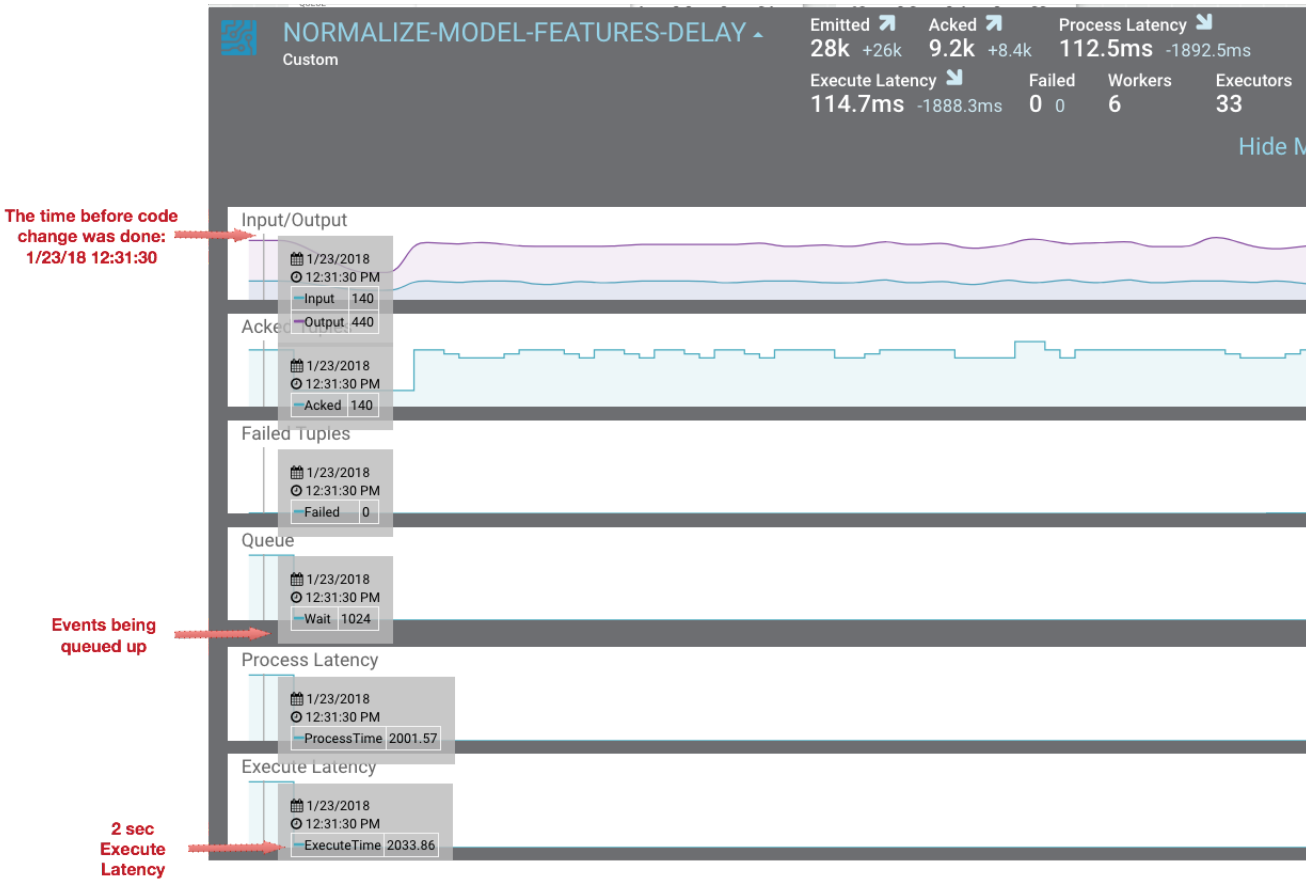
SAM's overview and detailed metrics makes it very easy to verify if the performance changes we made had the desired effect.

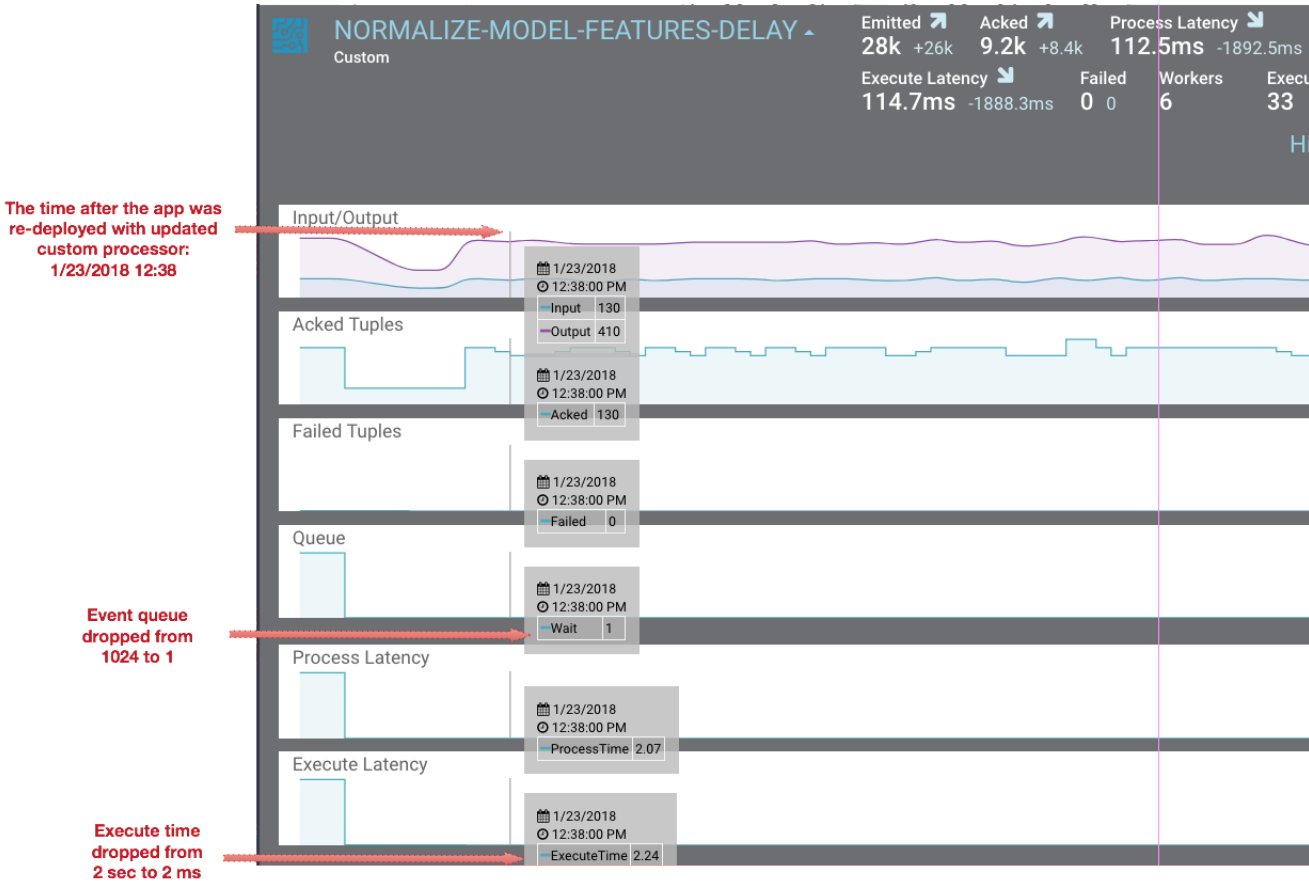
10.4.2.1. Latency Improvements

The below is the before and after metrics for the NORMALIZE-MODEL-FEATURES custom processor.

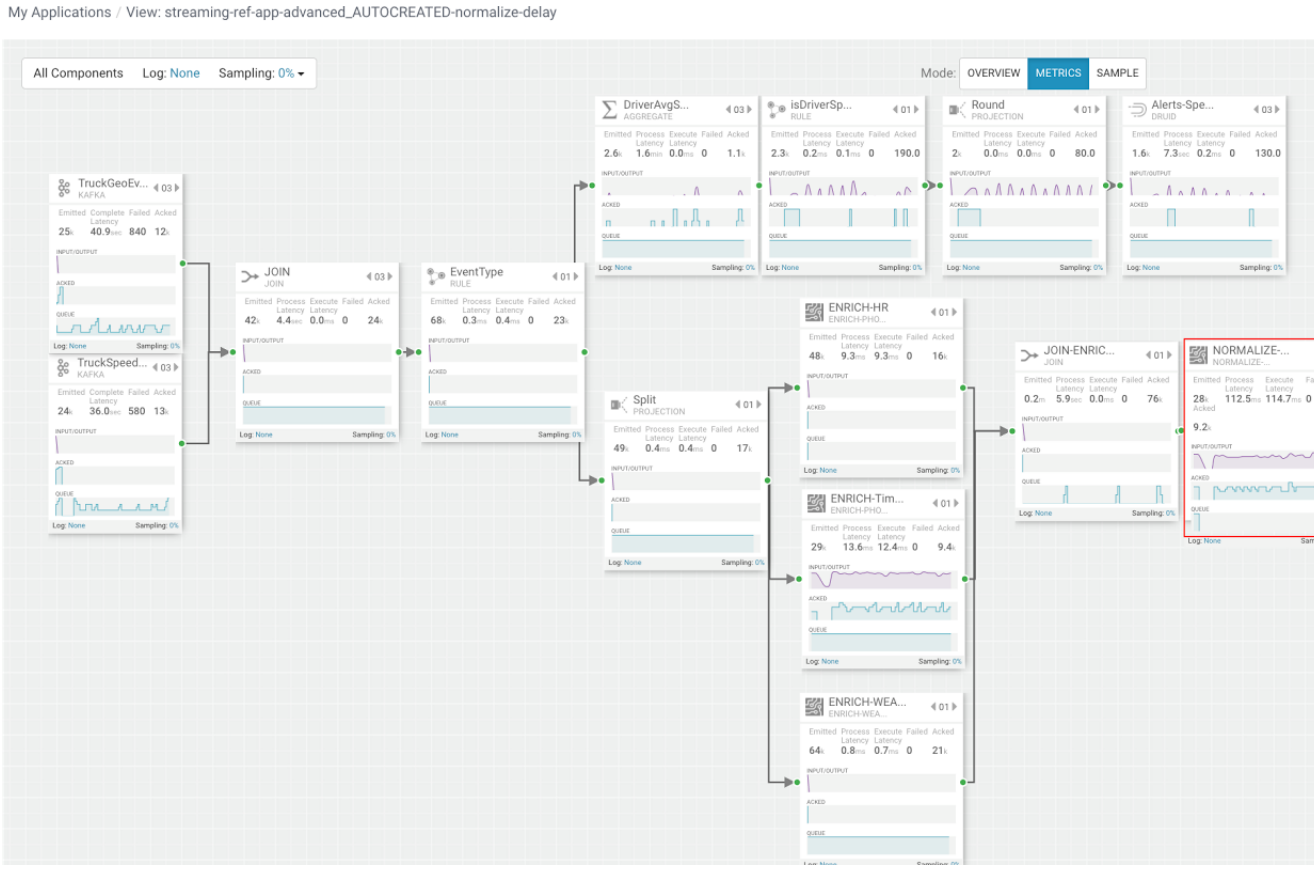


In the metric details view, the graphs provides an easy way to compare metrics before and after the code change.

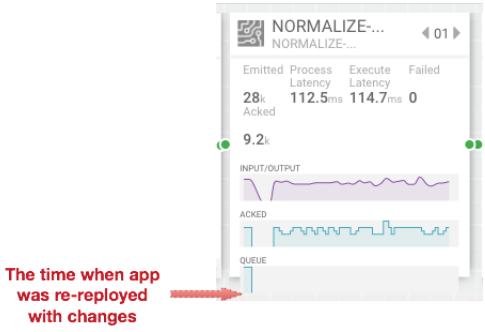

NORMALIZE-MODEL-FEATURES-DELAY
Custom
Emitted 28k +26k
Acked 9.2k +8.4k
Process Latency 112.5ms -1892.5ms
Execute Latency 114.7ms -1888.3ms
Failed 0 0
Workers 6
Executors 33
Hide M
The time before code change was done: 1/23/18 12:31:30
Input/Output
1/23/2018
12:31:30 PM
Input 140
Output 440
Acked Tuples
1/23/2018
12:31:30 PM
Acked 140
Failed Tuples
1/23/2018
12:31:30 PM
Failed 0
Queue
1/23/2018
12:31:30 PM
Wait 1024
Events being queued up
Process Latency
1/23/2018
12:31:30 PM
ProcessTime 2001.57
Execute Latency
1/23/2018
12:31:30 PM
ExecuteTime 2033.86
2 sec Execute Latency



You can also select the Metrics tab to validate the performance improvement.



If you zoom in on the NORMALIZE-MODEL-FEATURES component, you will see that after the code change is made, throughput increases and the wait drops to 0.



10.4.3. Debugging an Application through Distributed Log Search

About This Task

In a distributed system, searching for logs on different hosts for different components can be extremely tedious and time consuming. With SAM, all the application logs are indexed via the Ambari Log Search Server via Solr. SAM makes it easy to drill into and search for logs for specific components directly from the DAG view. Follow the below steps to use distributed log search:

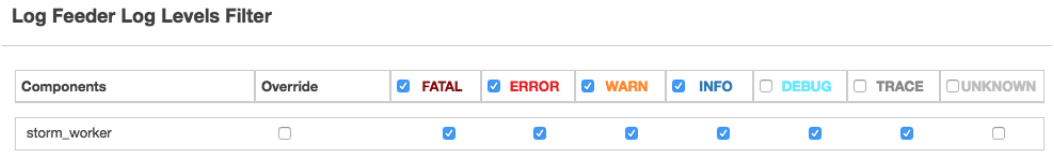
Steps

1. To enable Log Search in SAM, perform the following actions in Ambari.

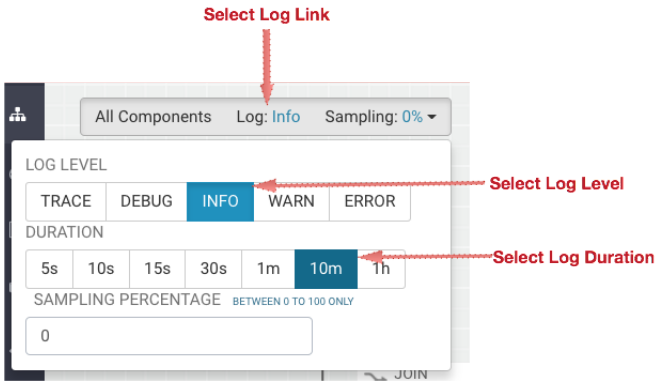
   a. In Ambari, select the Log Search service and select 'Log Search UI' from Quick Links

   b. Select the filter icon on the top right menu

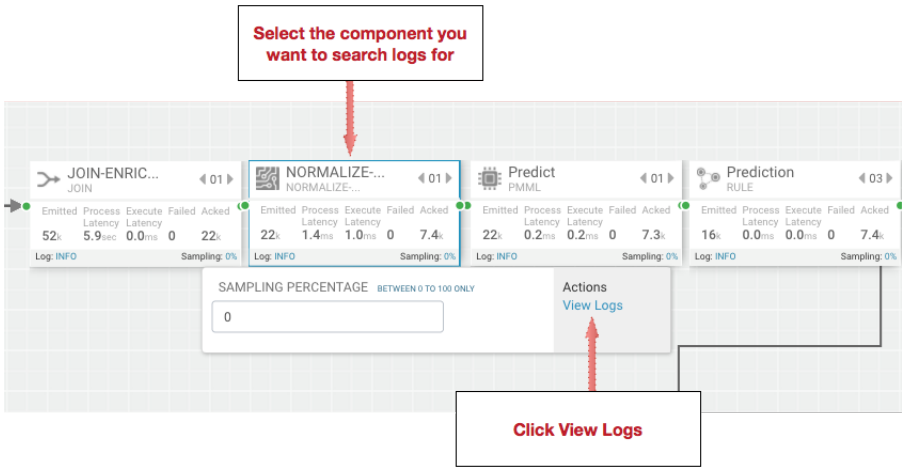
   c. For the storm_worker component, configure the filter like the following and click Save.



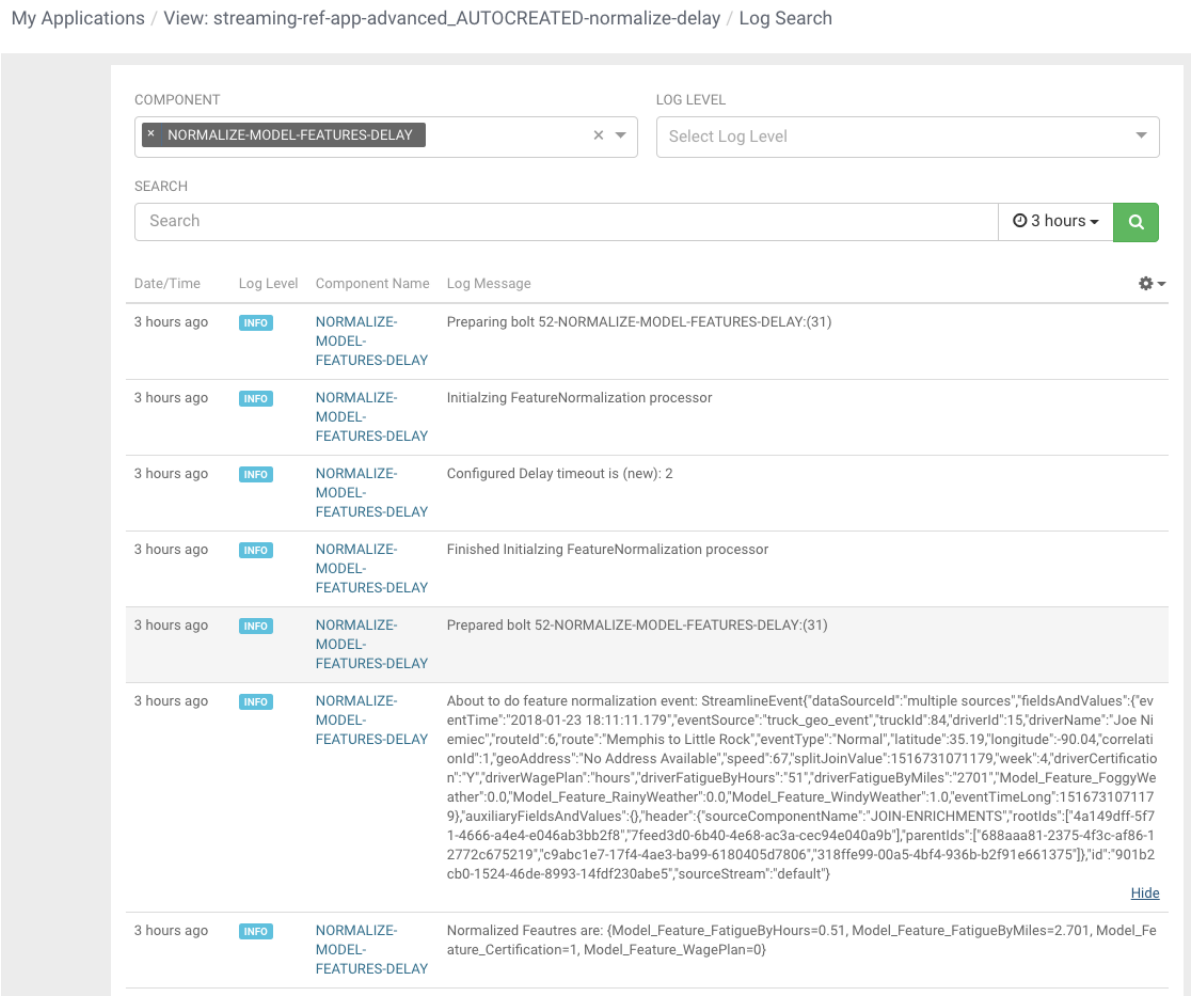
2. In SAM, you can dynamically change the logging level. For example, in SAM view mode of an application, click on the Log link, select the log level and the duration you want that log level.



3. Then click on the component you want to search logs for and under Actions select Logs.



4. This brings you to the Log Search page where you can search by component (s), log level(s) and search for strings using wildcard notation.



10.4.4. Debugging an Application through Sampling

About This Task

For troubleshooting, a convenient tool is to turn on sampling for a component or for the entire topology based on a sample percentage. Sampling allows you to log the emitted values of a component in the SAM App.

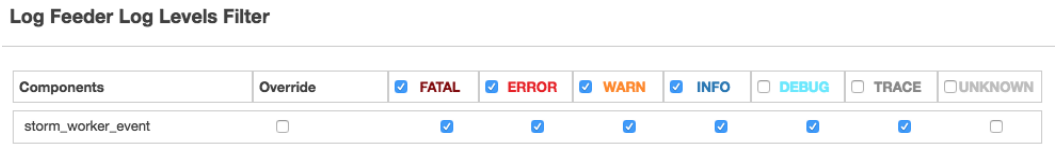
Steps

1. To enable Log Search in SAM, perform the following actions in Ambari.

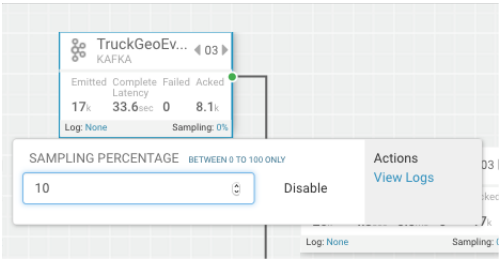
   a. In Ambari, select the Log Search service and select 'Log Search UI' from Quick Links

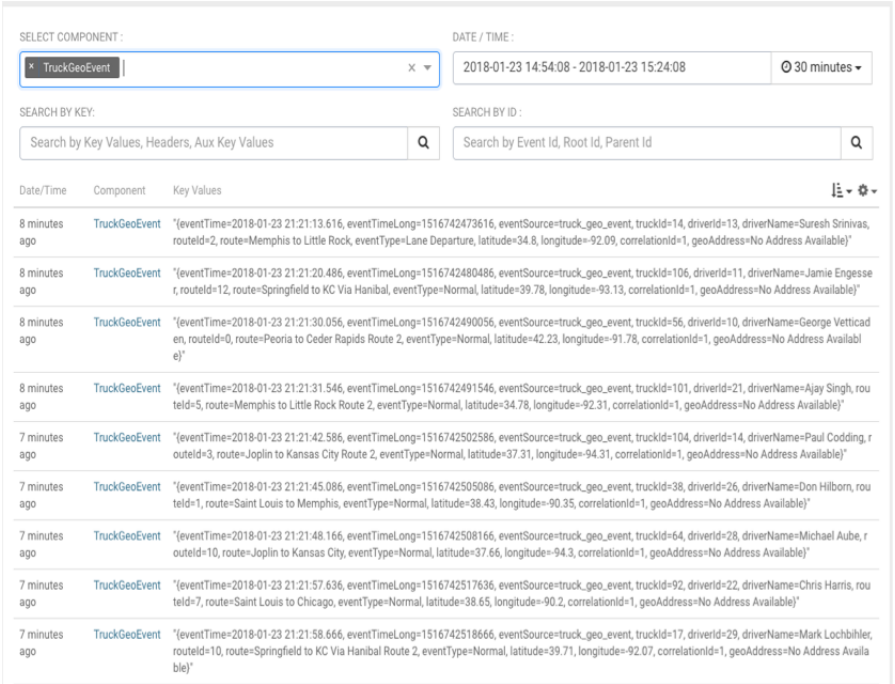
   b. Select the filter icon on the top right menu

   c. For the storm_worker_event component, configure the filter like the following and click Save.



2. In SAM view mode of the App, click on the component you want to turn on sampling for and enter a sampling percentage.



3. Click the 'SAMPLE' Tab .

4. Use the Sample Search UI to search for different events that were logged.