Hortonworks Streaming Analytics Manager 3

Getting Started with Streaming Analytics

Date of Publish: 2019-05-15

Hortonworks

https://docs.hortonworks.com/

Contents

Building an End-to-End Stream Application

A good way to get started using Hortonworks DataFlow (HDF) and all of its services like NiFi, Streams Messaging Manager, Streaming Analytics Manager, Schema Registry, etc 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 NiFi, SMM, SAM and Schema Registry and Spark Streaming. All the resources required to complete the tasks are provided in line.

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:

Procedure

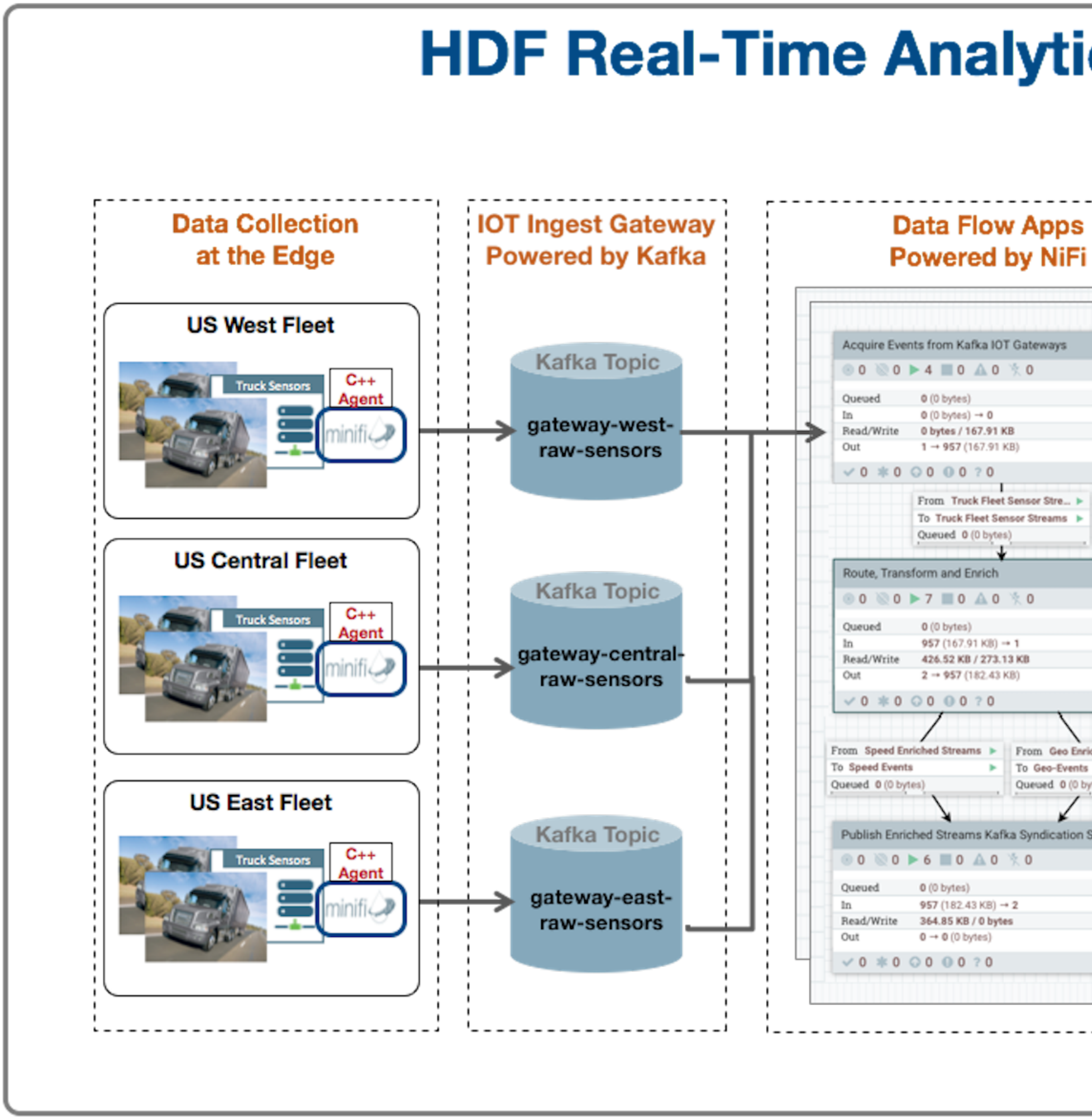
1. Outfit each truck with two sensors that emit event data such as timestamp, driver ID, truck ID, route, geographic location, and event type.
   - The geo event sensor emits geographic information (latitude and longitude coordinates) and events such as excessive braking or speeding.
   - The speed sensor emits the speed of the vehicle.
2. Stream the sensor events to an IoT gateway. The data producing app (e.g: a truck) will send CSV events from each sensor to one of three gateway topics (gateway-west-raw-sensors, gateway-east-raw-sensors or gateway-central-raw-sensors). Each event will pass the schema name for the event as a Kafka event header.
3. Use NiFi to consume the events from the Kafka topic, and then route, transform, enrich, and deliver the data from the gateways to two syndication topics (e.g: syndicate-geo-event-avro, syndicate-speed-event-avro, syndicate-geo-event-json, syndicate-speed-event-json) that various downstream analytics applications can subscribe to.
4. Connect to the two streams of data to perform analytics on the stream.
5. Join the two sensor streams using attributes in real-time. For example, join the geo-location stream of a truck with the speed stream of a driver.
6. Filter the stream on only events that are infractions or violations.
7. All infraction events need to be available for descriptive analytics (dash-boarding, visualizations, or similar) by a business analyst. The analyst needs the ability to perform analysis on the streaming data.
8. Detect complex patterns in real-time. For example, over a three-minute period, detect if the average speed of a driver is more than 80 miles per hour on routes known to be dangerous.
9. When each of the preceding rules fires, create alerts, and make them instantly accessible.
10. Execute a logistical regression Spark ML model on the events in the stream to predict if a driver is going to commit a violation. If violation is predicted, then generate an alert.
11. Monitor and manage the entire application using Streams Messaging Manager and Stream Operations.

What to do next

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.

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.



Prepare Your Environment

Deploying Your Cluster

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.

Procedure

1. Install Ambari 2.7.0.
2. Install HDP 3.0.0.
3. Install the HDF 3.2.0 Management Pack onto the HDP cluster.

What to do next

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 Matrices*.

Registering Schemas in Schema Registry

The trucking application streams CSV events from the truck sensors to a IOT gateway powered by a set of Kafka gateway topics. NiFi consumes the events from these topics, and then routes, enriches, and delivers them to a set of syndication Kafka topics.

To do this, you must perform the following tasks:

- Create the Kafka gateway and syndication topics
- Register a set of Schemas in Schema Registry

Create the Kafka Topics

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

Procedure

1. Log into the node where Kafka broker is running.
2. Download script createKafkaTopics.sh from here: https://raw.githubusercontent.com/georgevetticaden/sam-trucking-data-utils/hdf-3-2/src/main/resources/scripts/smm/createKafkaTopics.sh
3. Execute the script by passing as arg ZK:Port

   For example:

   ```
   ./createKafkaTopics.sh ZK_HOST:2181
   ```

Register Schemas for the Kafka Topics

Register schemas for the IOT gateway and syndication Kafka topics .Registering the Kafka topic schema 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.

Procedure

1. Log into the node where you have access to the Schema Registry Server.
2. Download the Data-Loader and unzip the contents.

3. Navigate to the Data-Loader directory:

```
cd Data-Loader
```

4. Execute the following:

```
java -cp \
stream-simulator-jar-with-dependencies.jar \
hortonworks.hdf.sam.refapp.trucking.simulator.schemaregistry.TruckSchemaRegistryLoade
 \
$SCHEMA_REGISTRY_ENDPOINT_URL

E.g: SCHEMA_REGISTRY_ENDPOINT_URL = http://SR_HOST::7788/api/v1
```

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

To prepare to perform 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/custom-processor/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 HDFS 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:

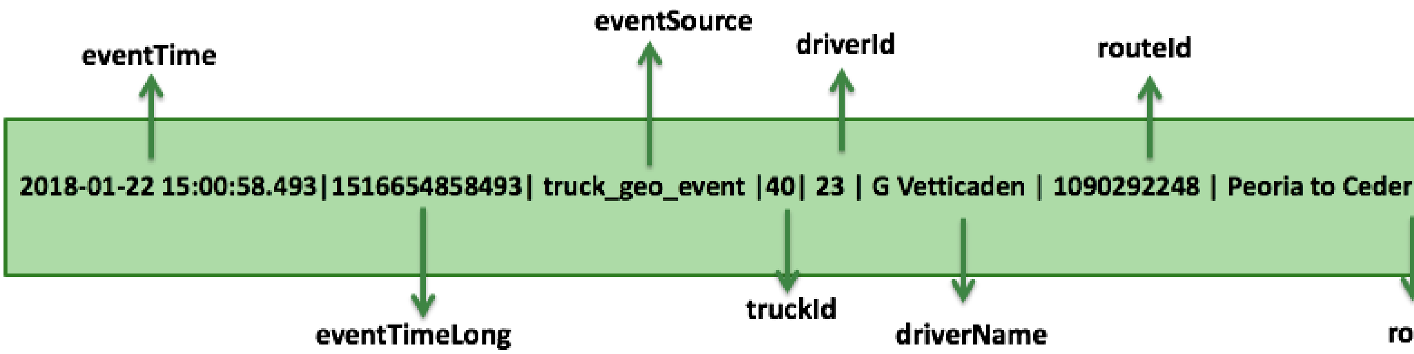
```
su hdfs
hadoop fs -mkdir /apps/trucking-app
hadoop fs -chmod 777 /apps/trucking-app
```

Creating a Dataflow Application

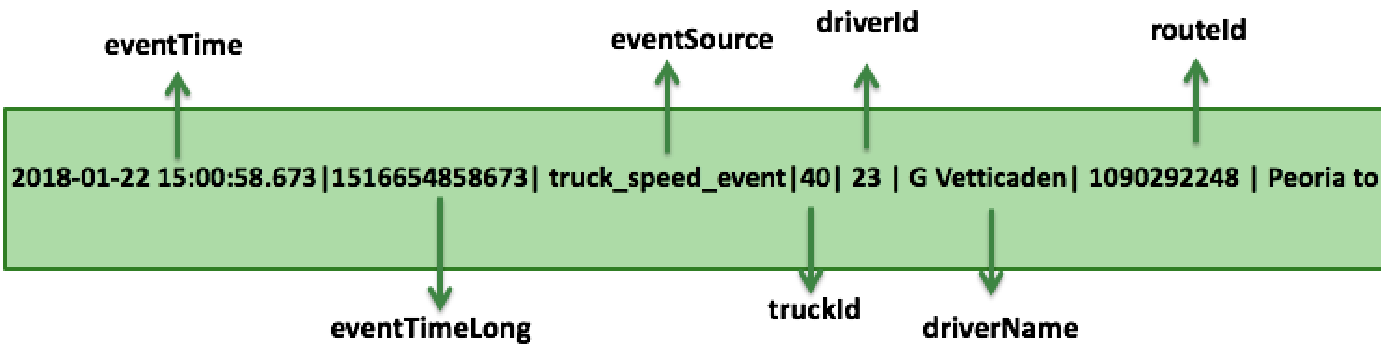
Data Producer Application Generates Events

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

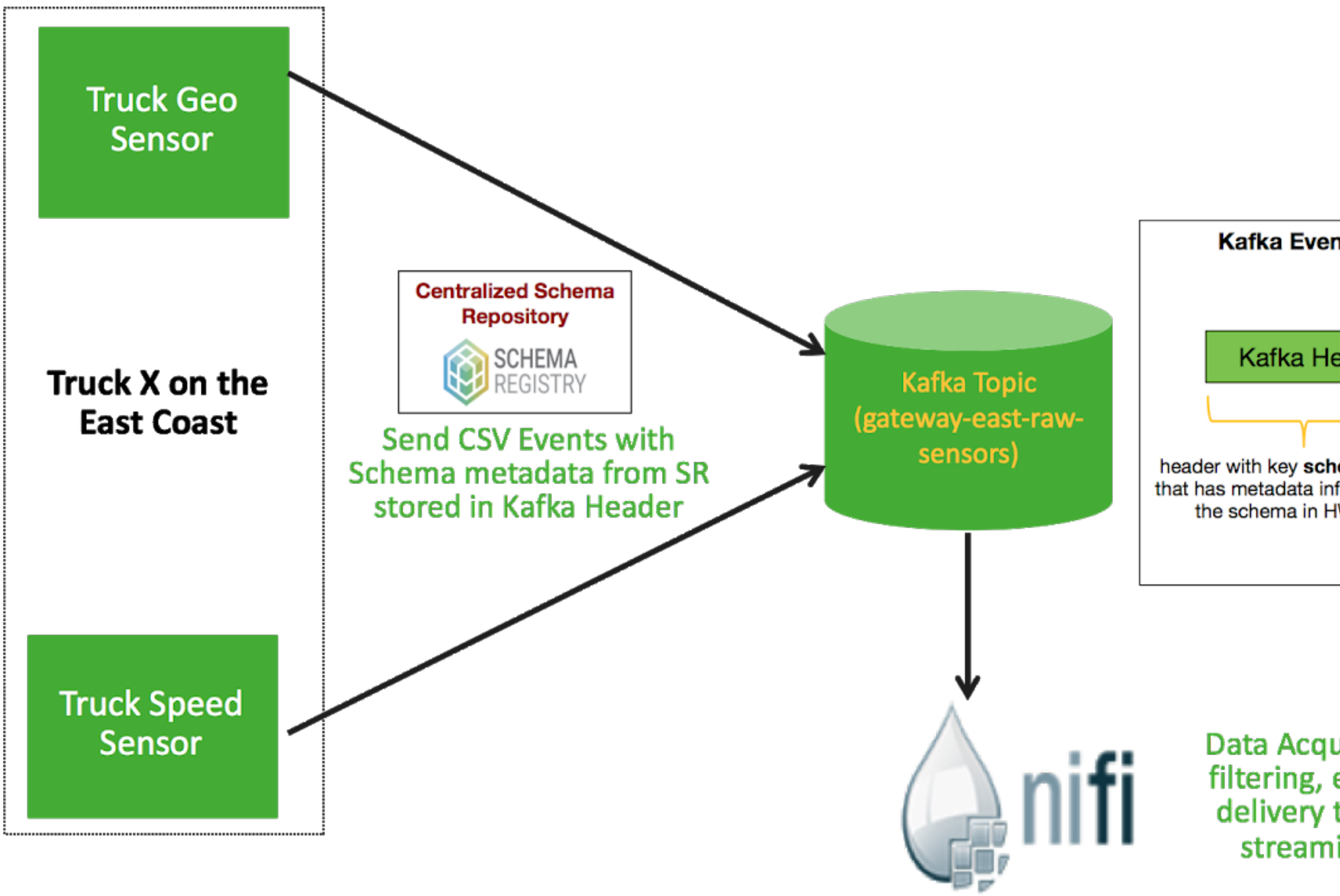
Geo Sensor Stream



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:

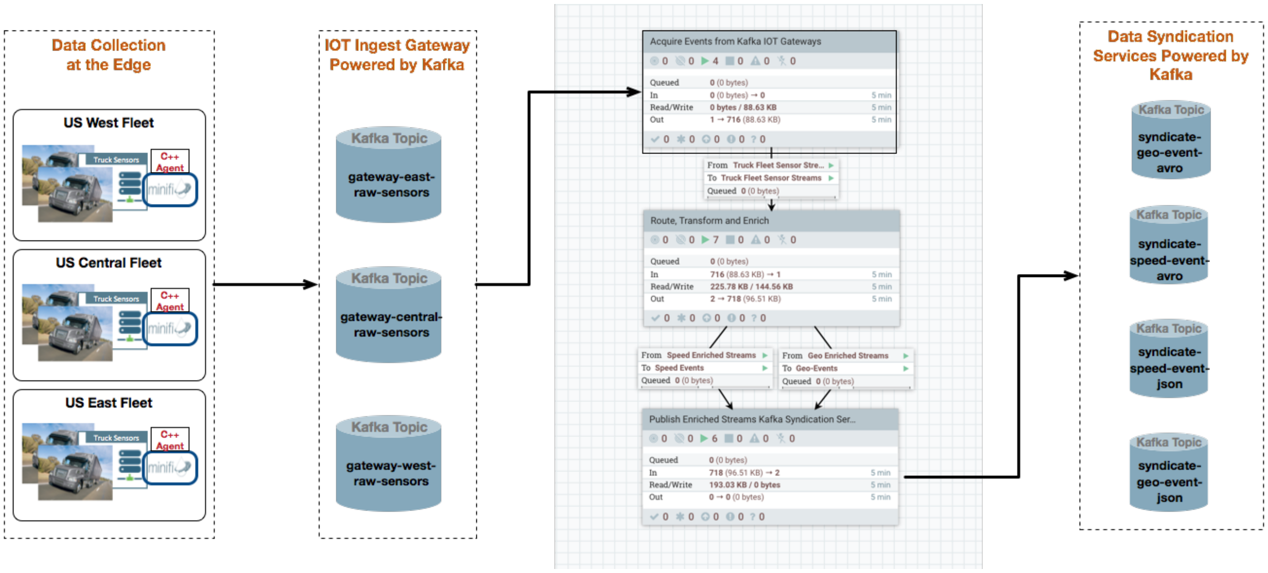


Use NiFi's Kafka 1.X ConsumeKafkaRecord and PublishKafkaRecord processors using record-based processing to do the following:

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
5. Filter, Enrich, Transform

6. Deliver the data to downstream syndication topics

The below diagram illustrates the entire flow and the subsequent sections walks through how to setup this data flow.



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 IOT Trucking Fleet - Data Flow process group. The following instructions are with respect to that flow.

NiFi Controller Services

Procedure

1. Click on Flow Configuration Settings icon, select Controller Services tab, and select Hortonworks Schema Registry Controller Service.
2. Click on Flow Configuration Settings icon and select **Controller Services** tab.

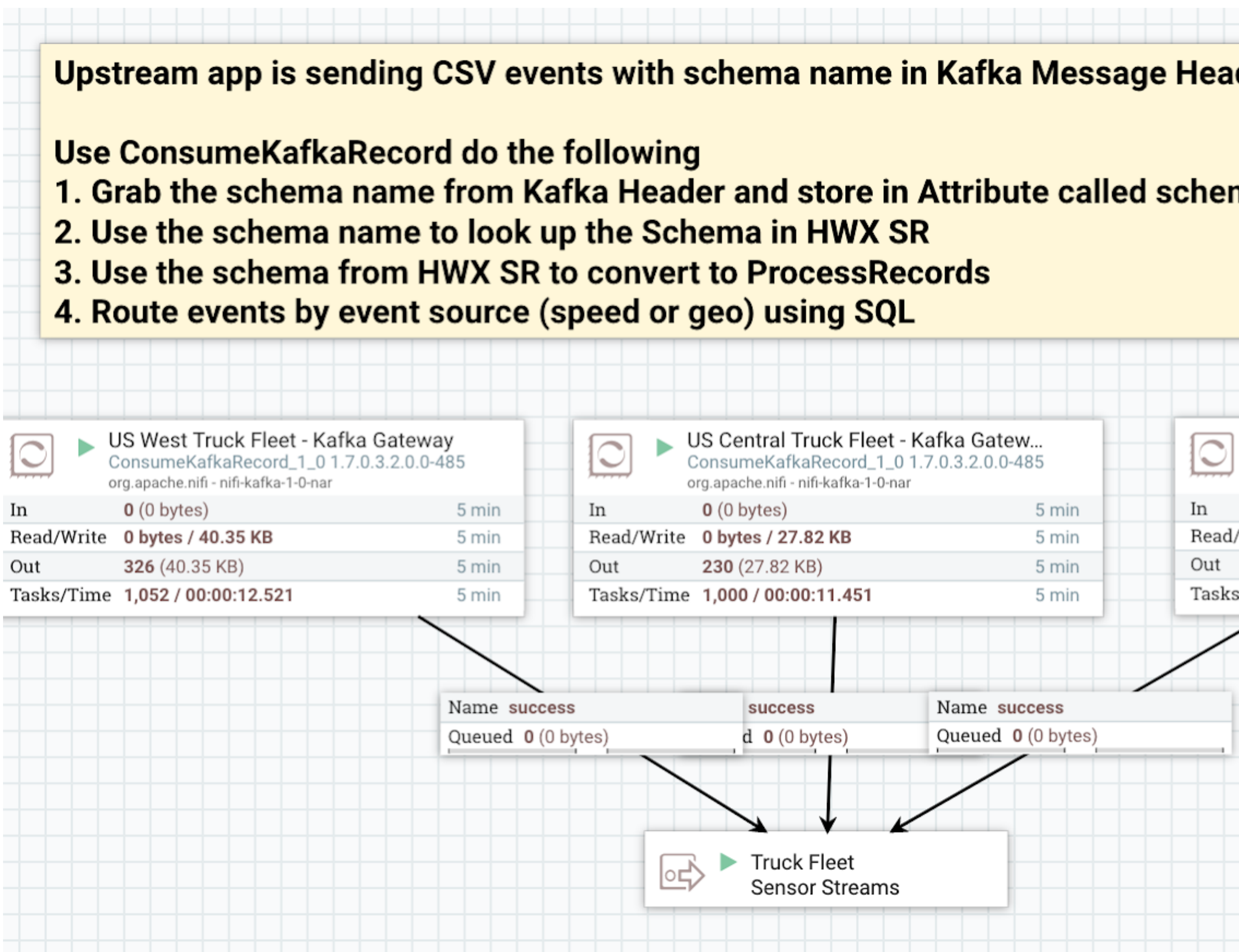
   You will see the HWX Schema Registry controller service. Edit the properties to configure the Schema Registry URL based on your environment. You can find this value in the Streaming Analytics Manager Service in Ambari for the configuration property called registry.url. An example of what the URL looks similar to http://$REGISTRY_SERVER:7788/api/v1.
3. 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.
4. Enable the HWX Schema Registry and the other controller services. You should have 5 controller services enabled like the following.

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

NiFi Ingests the Raw Sensor Events

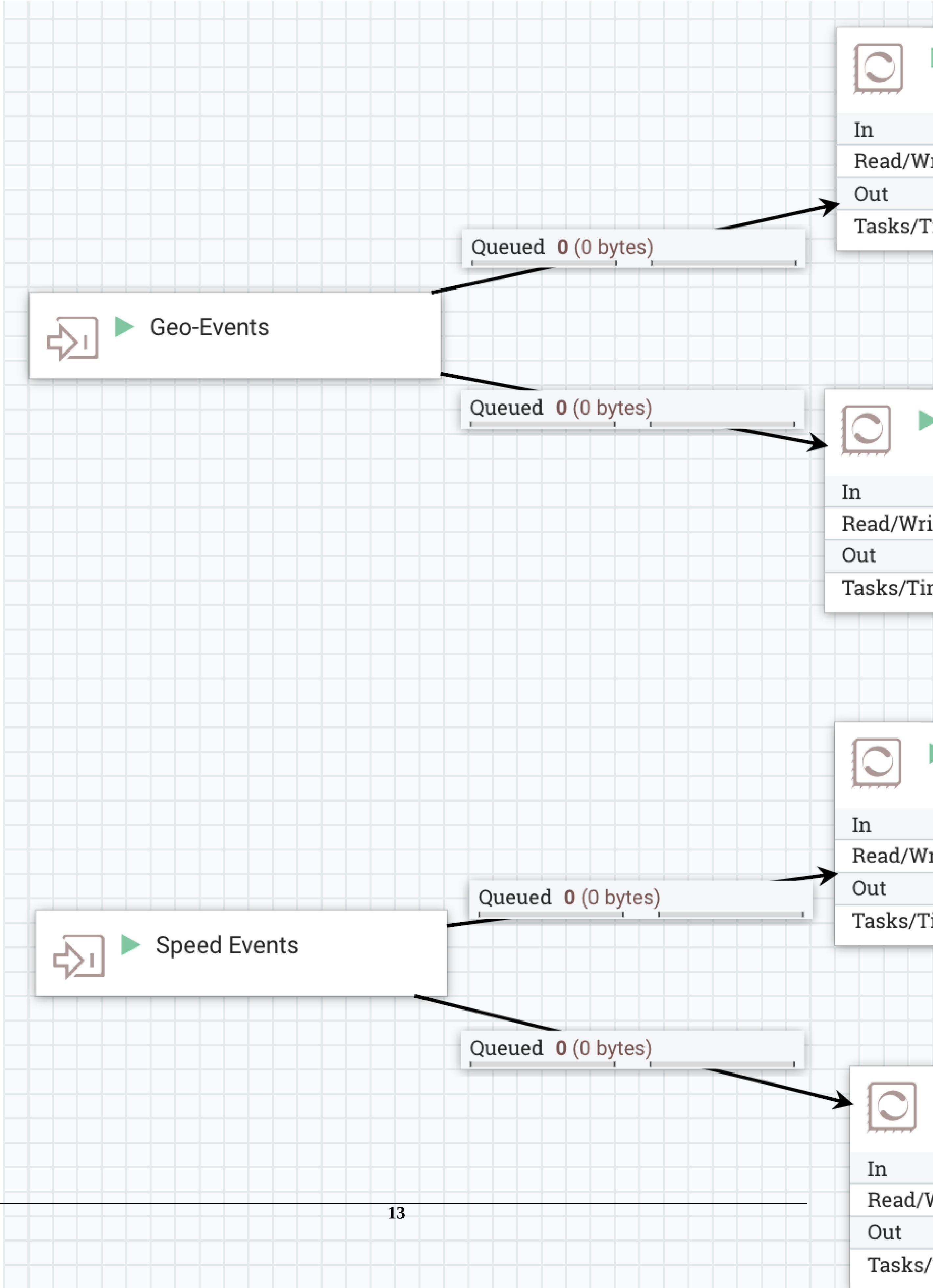
In the IOT Trucking Fleet - Data Flow process group, go into the "Acquire Events from Kafka IOT Gateways" process group. The first step in the NiFi flow is to ingest the csv events from the three IOT gateway topics. We will use the new Kafka 1.0 ConsumerKafkaRecord processor for this.



Configure the ‘Kafka Brokers’ value for each of the three ConsumeKafkaRecord_1_0 processors based on your cluster.

Publish Enriched Events to Kafka for Consumption by Analytics Applications

After NiFi completes the routing, transforms, and enrichment, NiFi publishes the enriched events into a set of syndication Kafka topics. This is done in the Publish Enriched Streams Kafka Syndication Services process group. This process group uses the PublishKafkaRecord processor to publish the events into 4 syndication topics.


Geo-Events
Speed Events
Queued 0 (0 bytes)
Queued 0 (0 bytes)
Queued 0 (0 bytes)
Queued 0 (0 bytes)
In
Out
In
Out
In
Out
In
Out

Ensure that for the PublishKafkaRecord, you change the Kafka Brokers property value to your cluster settings.

Start the NiFi Flow

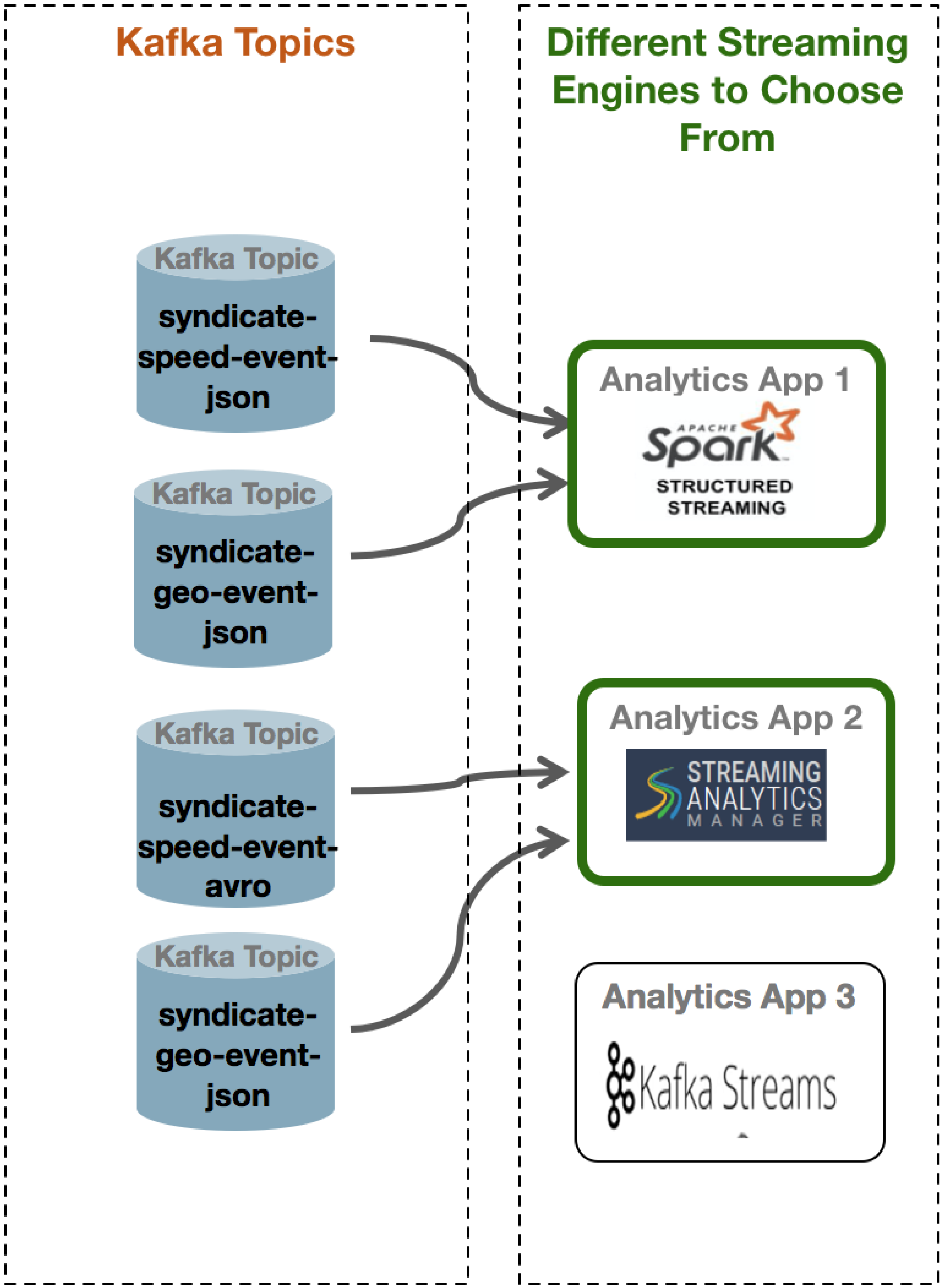
Start the Process Group called IOT Trucking Fleet - Data Flow.

Pick your Streaming Engine

Hortonworks supports a number of powerful Streaming Engines including:

- Spark Streaming (Currently Supported in HDP)
- Streaming Analytics Manager (SAM) using Apache Storm (Currently Supported in HDF)
- Apache Kafka Streams (Will be supported in future HDP/HDF release)

Hortonworks provides you the flexibility to pick the streaming engine of your choice to build streaming analytics application.


Kafka Topics
Different Streaming Engines to Choose From
Kafka Topic
syndicate-speed-event-json
Kafka Topic
syndicate-geo-event-json
Kafka Topic
syndicate-speed-event-avro
Kafka Topic
syndicate-geo-event-json
Analytics App 1
APACHE Spark
STRUCTURED STREAMING
Analytics App 2
STREAMING ANALYTICS MANAGER
Analytics App 3
Kafka Streams

If your organization has not standardized on a streaming engine and is looking for guidance on choosing an engine, use the below table to help guide your selection.

Requirement	Streaming Engine to Use
You want to build streaming applications with as little code as possible and want to use ETL like Tooling with a drag and drop paradigm to build streaming apps	Streaming Analytics Manager (SAM) with Storm
You want the ability to build an app that you can deploy in batch and/ or streaming mode	Spark Streaming
You plan to develop streaming applications using scala and java and want a clean easy to use API	Spark Streaming
You want the ability to execute SQL against the stream	Spark Streaming
You want process event one at a time (no microbatching)	Streaming Analytics Manager (SAM) with Storm

The below sections walk you through implementing the streaming analytics requirements with these different tools/ engines.

Creating a Streaming Analytics Application with SAM

Creating a Stream Analytics Application with SAM

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.

Setting up the Stream Analytics App using the TruckingRefAppEnvEnviornmentBuilder

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.

Procedure

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.hdp.ambari.url = The rest endpoint for the HDP/HDF cluster you installed (e.g.: http:/<HDP_AMBARI_HOST>:<PORT>/api/v1/clusters/<cluster_name>)
   - sam.service.pool.hdp.ambari.username = the username to log into Ambari
   - sam.service.pool.hdp.ambari.passwd = the password to log into Ambari
   - 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.SMMTruckingRefAppEnviornmentBuilderImpl
trucking-advanced-ref-app.properties
```

Results

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
```

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 reference application:

Procedure

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.

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.

Creating Your First Application

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**.

About this task

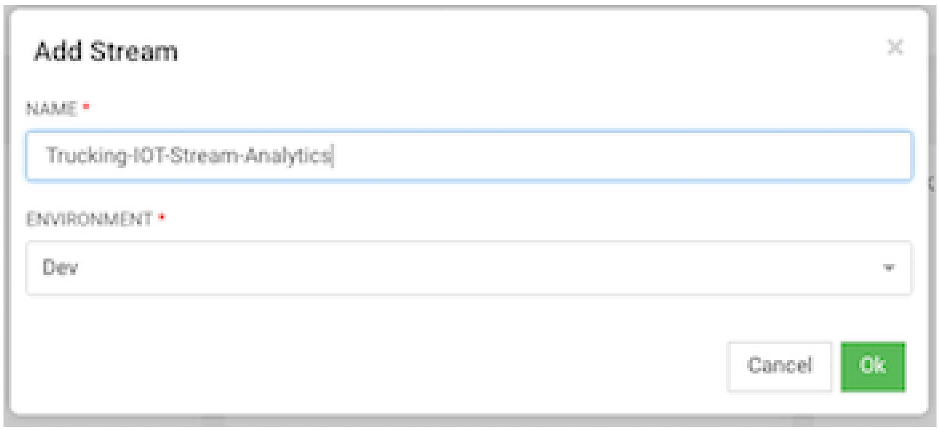
Creating a new stream application requires two steps: clicking the **+** icon, and then providing a unique name for the stream application and associating the application with an Environment.

Procedure

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.



Results

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.

Creating and Configuring the Kafka Source Stream

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.

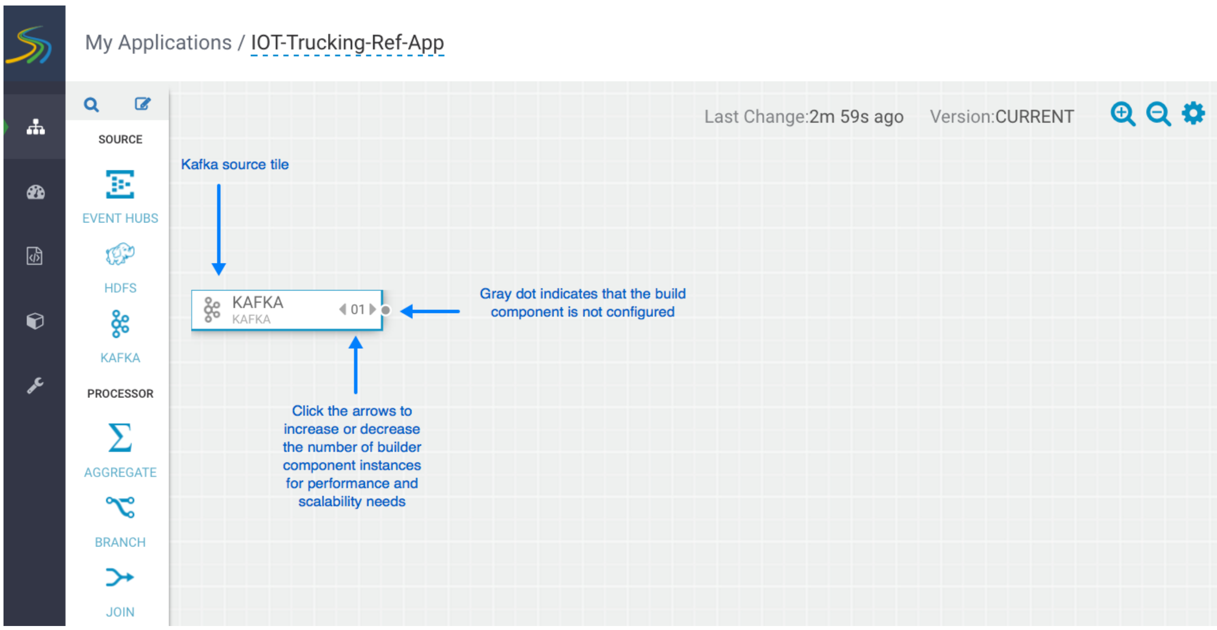
About this task

Every stream application must start with a source.

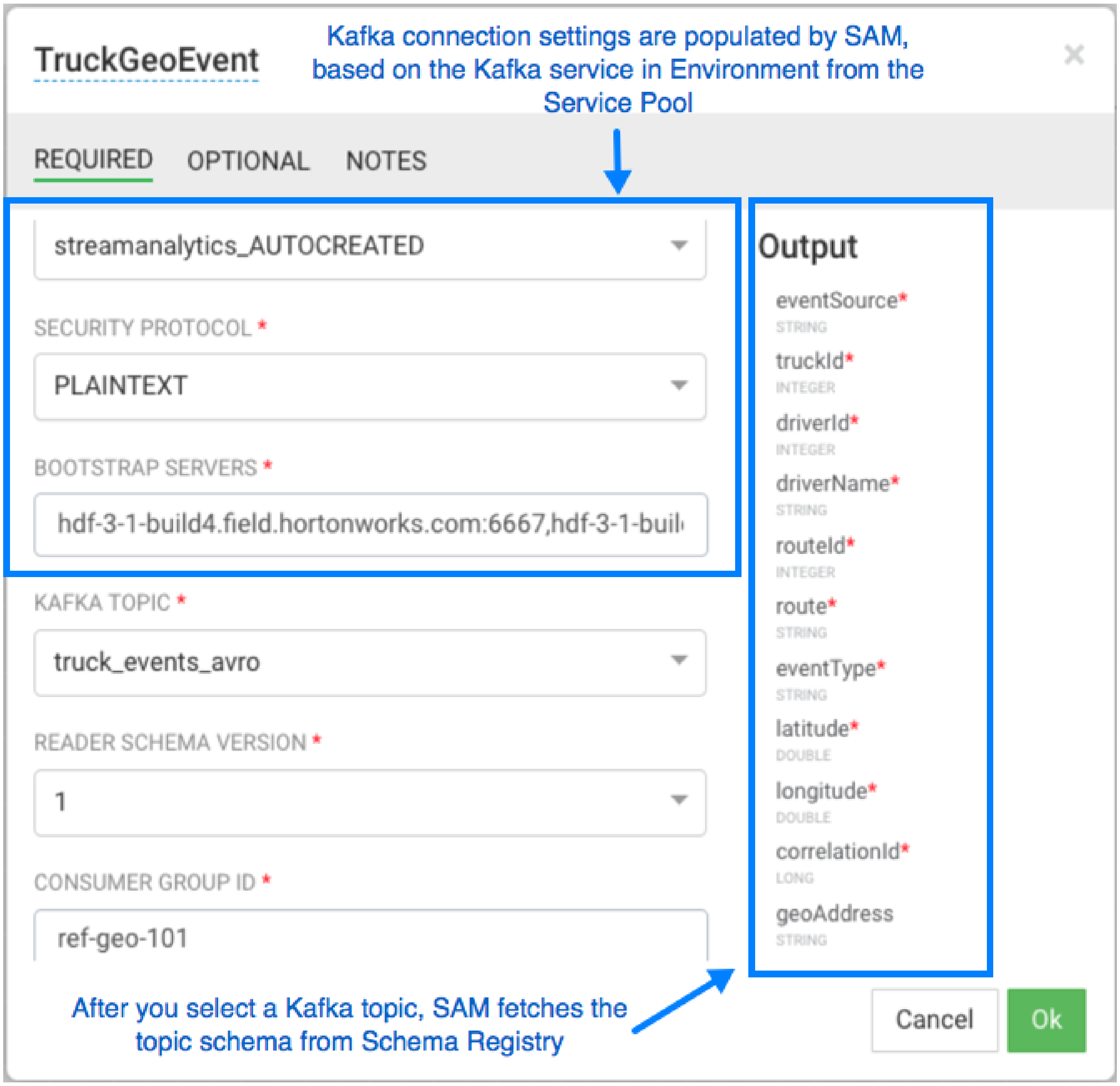
Complete the following instructions to start building a stream application. Use these steps to implement Requirement 4 of the use case.

Procedure

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:



Results

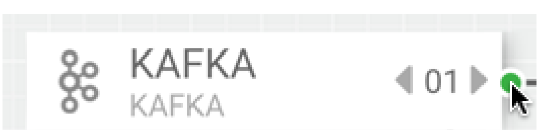
Once you have configured your Kafka component correctly, the tile component displays a green dot.

Connecting Components

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.

Procedure

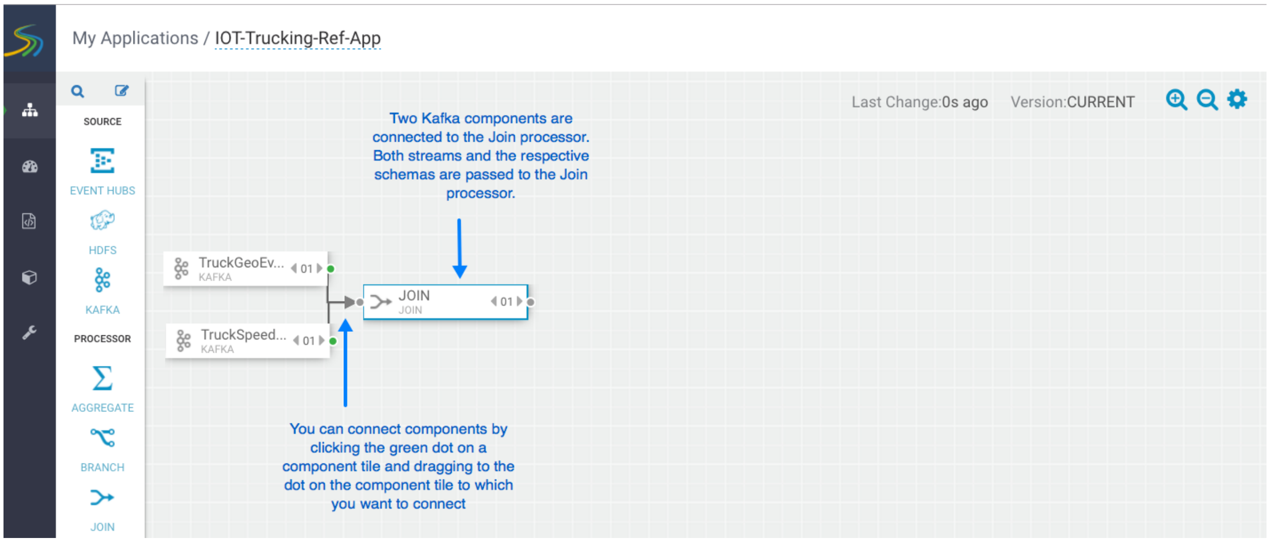
1. Click the green dot to the right of your source component.



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.

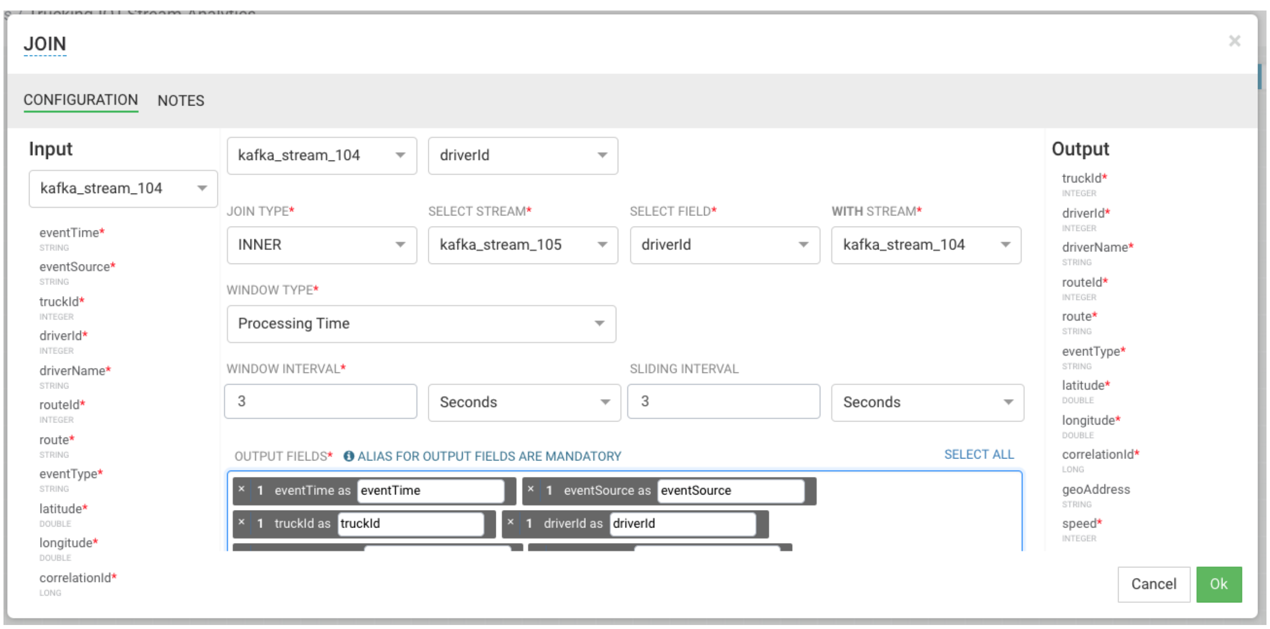


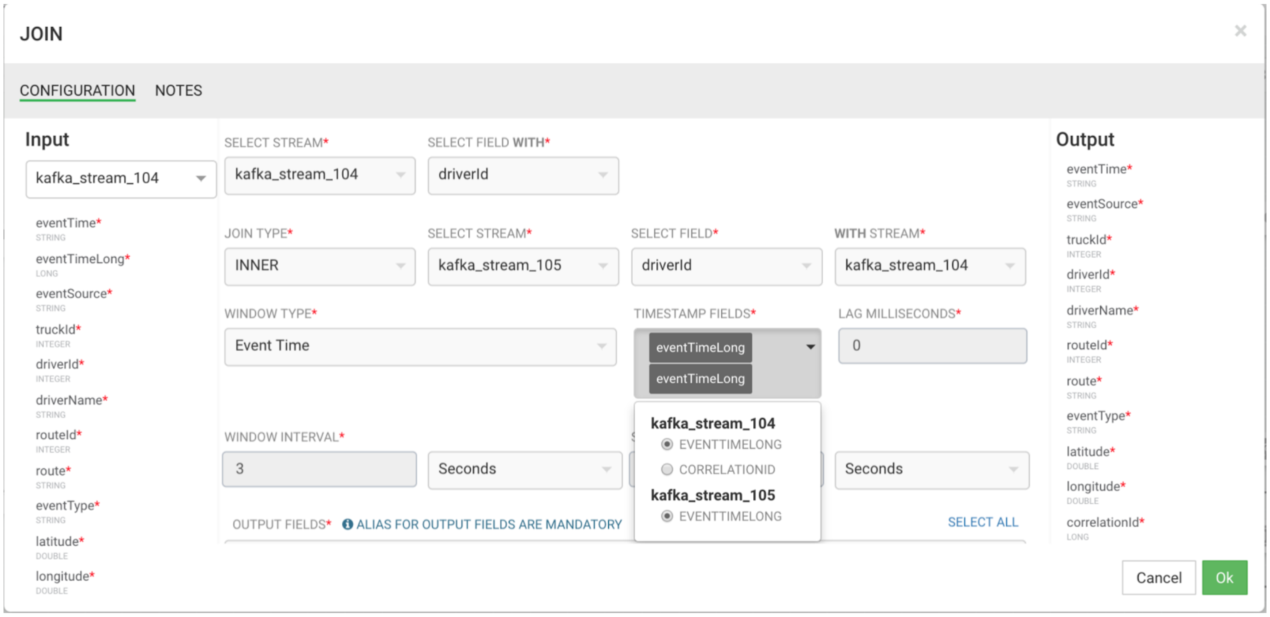
Joining Multiple Streams

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:

About this task

- 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.





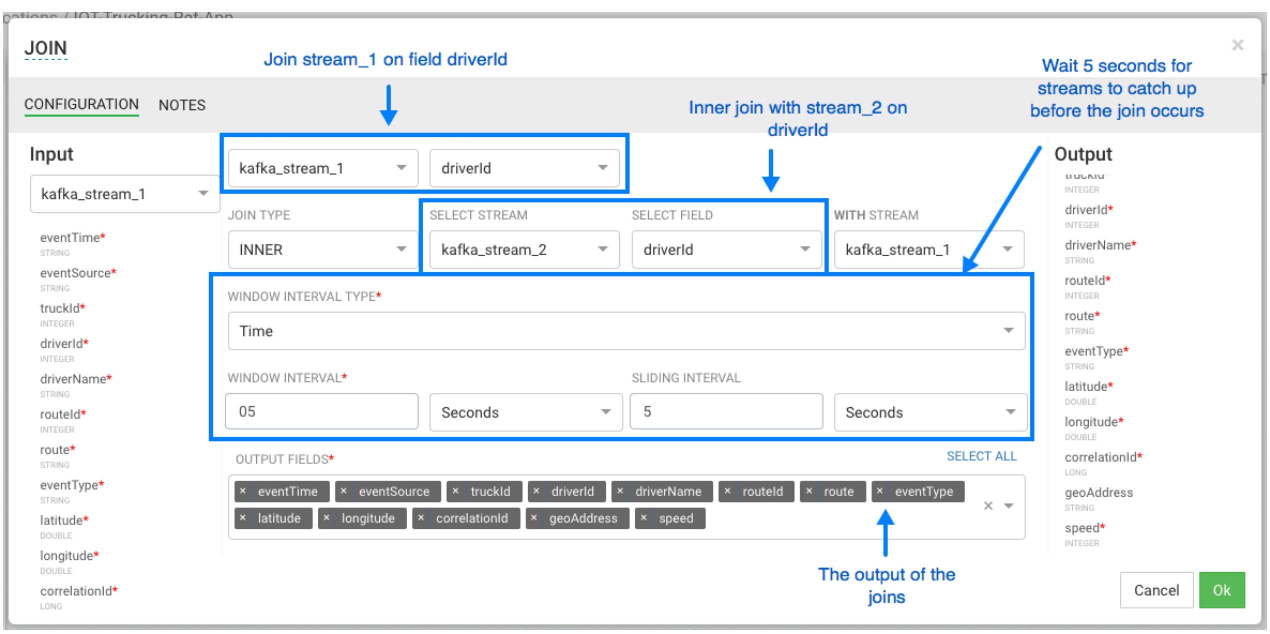
For the purposes of this reference application, use processing time.

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.

Procedure

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



Filtering Events in a Stream using Rules

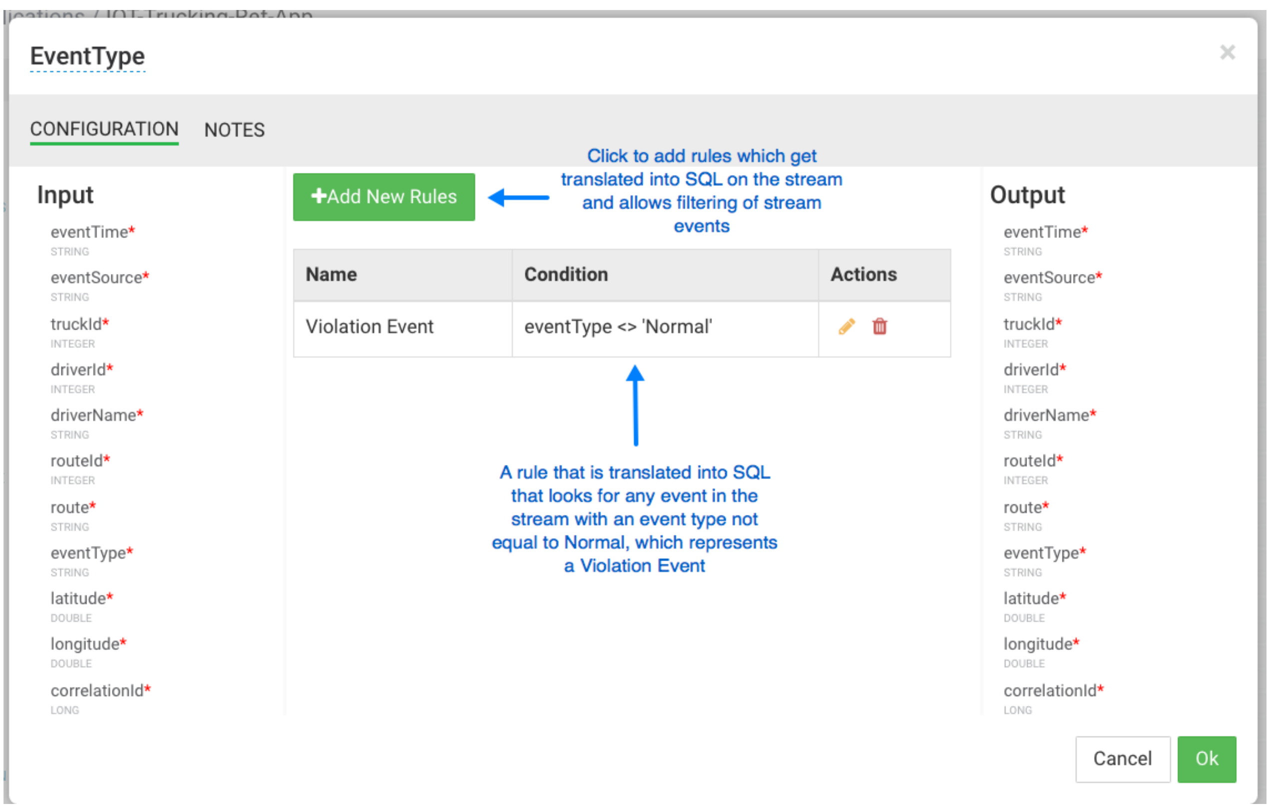
SAM provides powerful capabilities to filter events in the stream. It uses a Rules Engine, which translates rules into SQL queries that operate on the stream of data. The following steps demonstrate this, implementing Requirement 6 of the use case.

Procedure

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

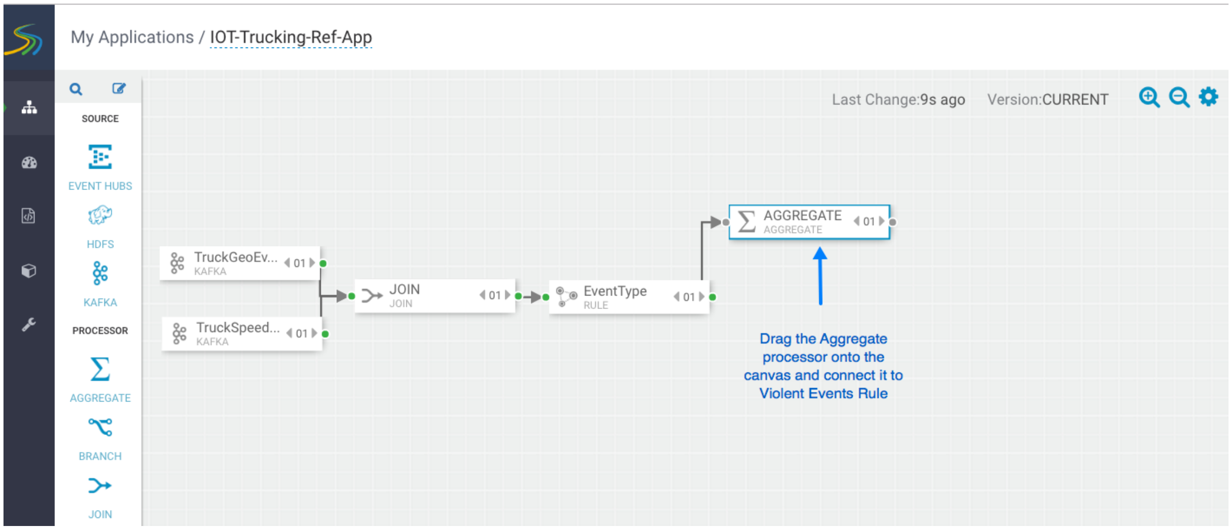


Using Aggregate Functions over Windows

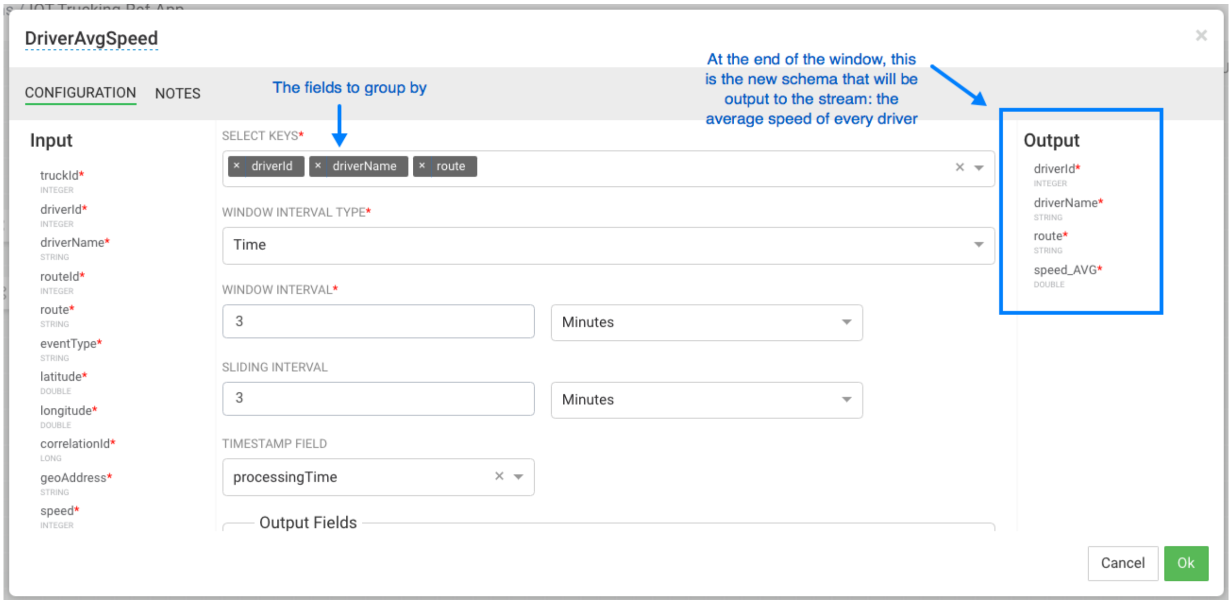
Windowing is the ability to split an unbounded stream of data into finite sets based on specified criteria such as time or count, so that you can perform aggregate functions (such as sum or average) on the bounded set of events. SAM exposes these capabilities using the Aggregate processor. The Aggregate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

Procedure

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.

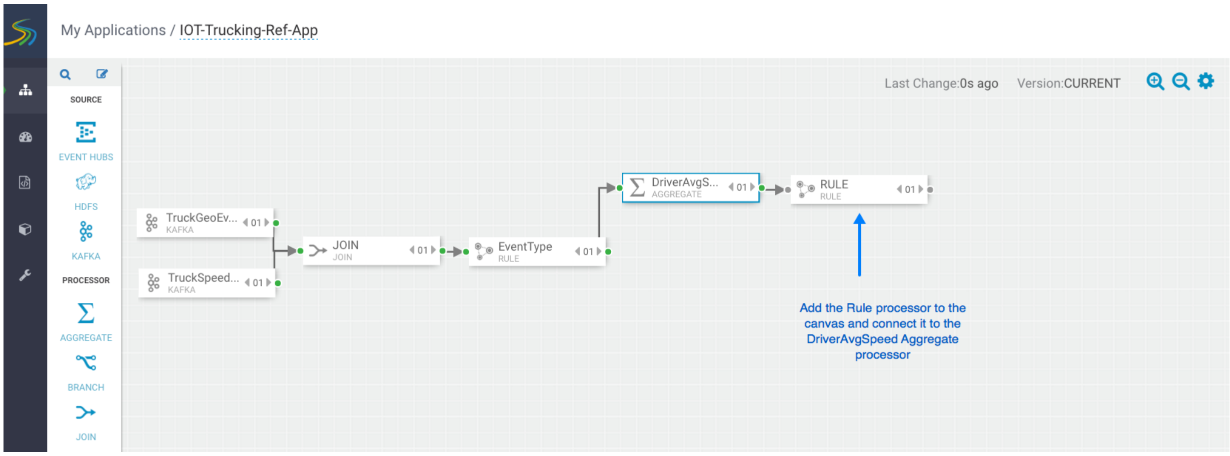


Implementing Business Rules on the Stream

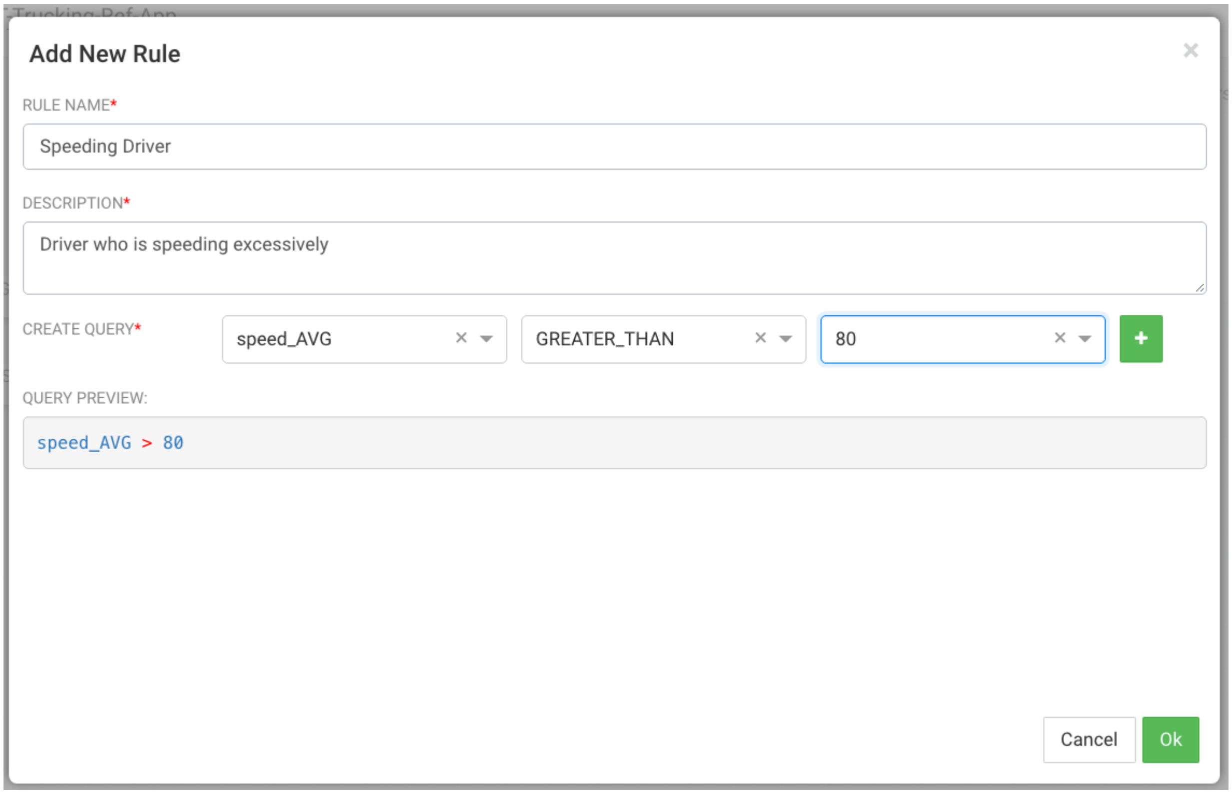
This section shows you how to implement the business rule you created above to detect high speeding drivers. "High speed" is defined as greater than 80 miles per hour over a three-minute time window. This step partially implements Requirement 8 of the use case.

Procedure

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

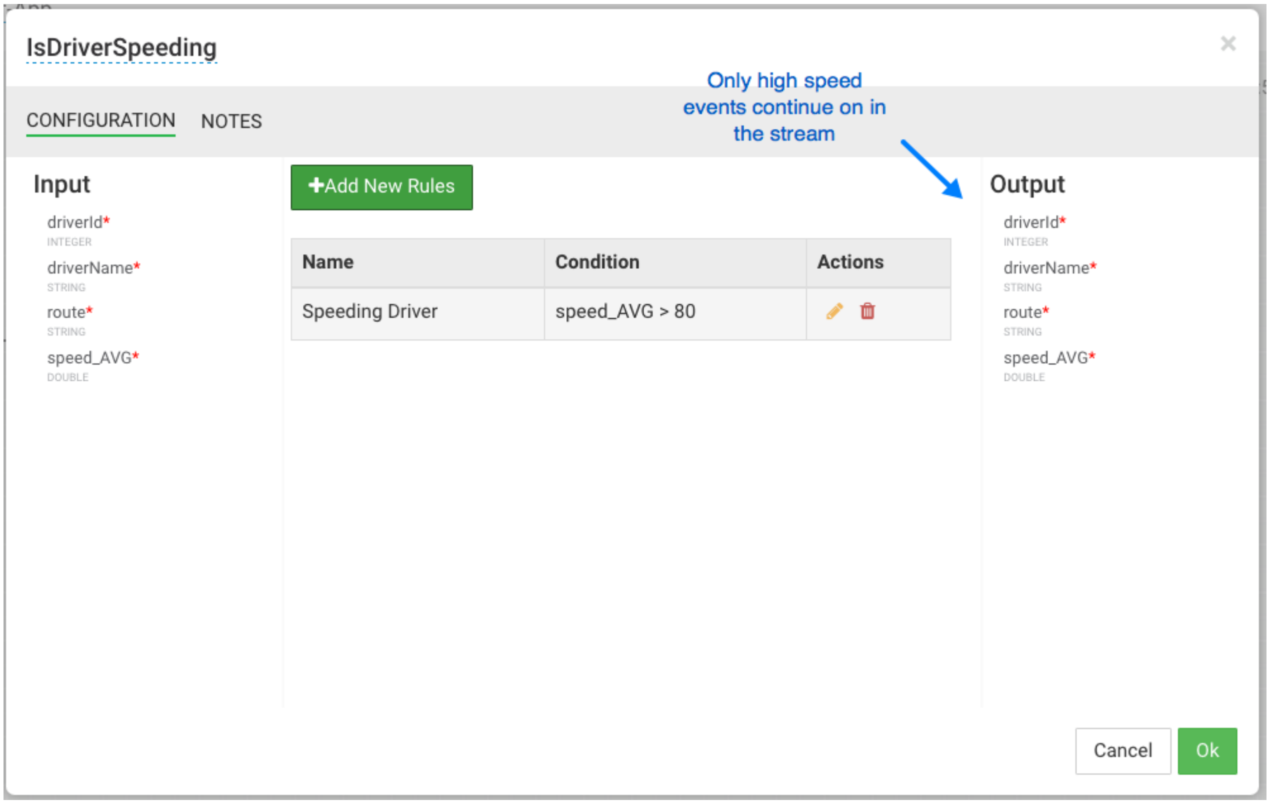


2. Configure the business rule as follows:



Results

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

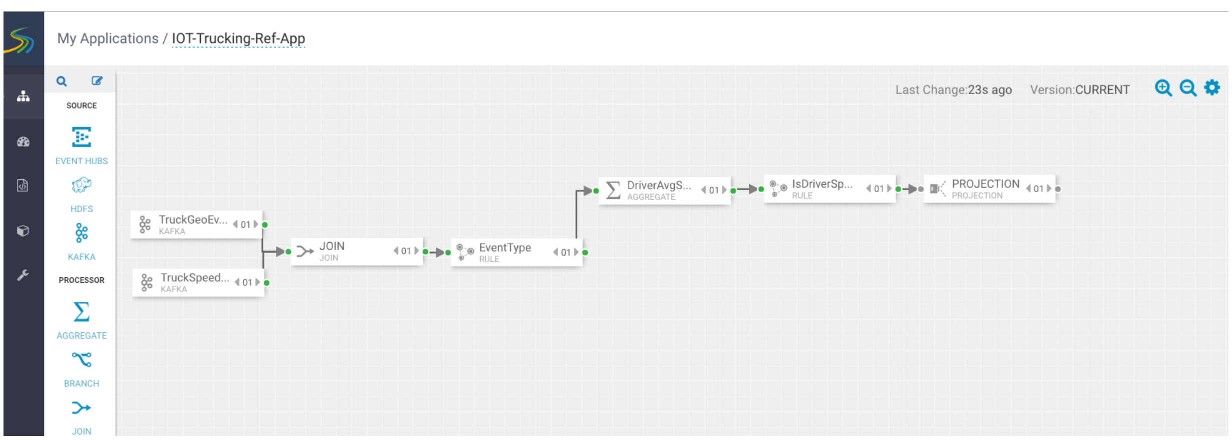


Transforming Data using a Projection Processor

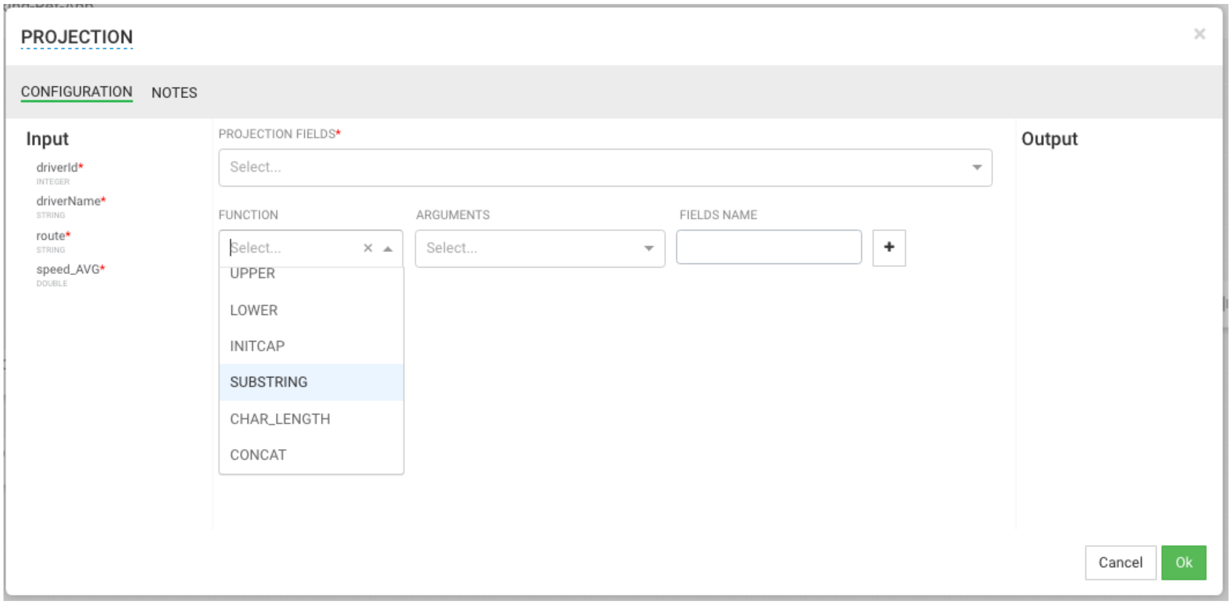
It is common to do transformations on the events in the stream. In our case, before we alert on the speeding driver, we want to convert the average speed we calculated in the aggregate processor into a integer from a double so it is easier to display in the alert. The projection processor allows you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

Procedure

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



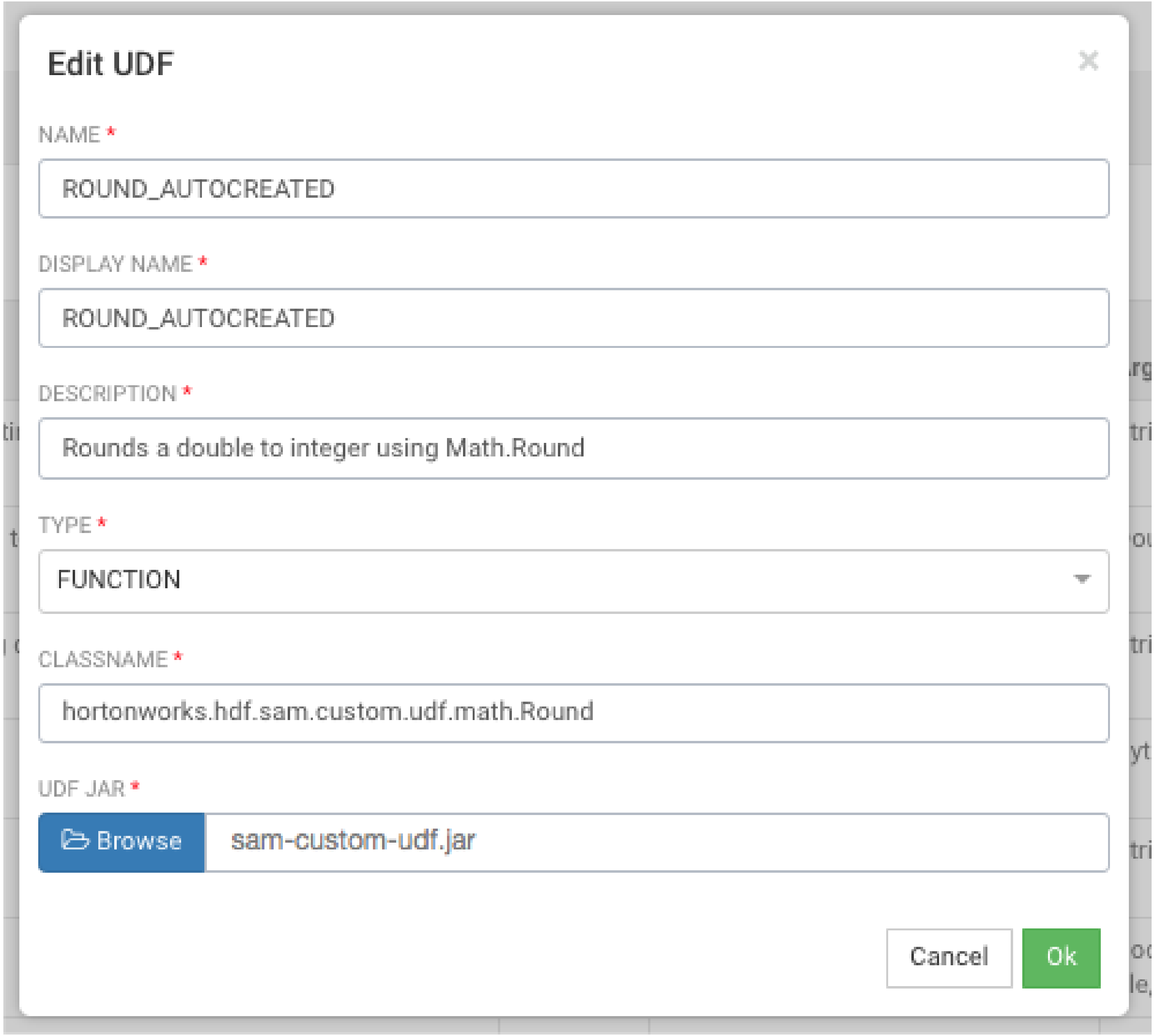
2. When you double-click on the projection processor, you see a number of out-of-the-box functions, however a Round function does not exist.



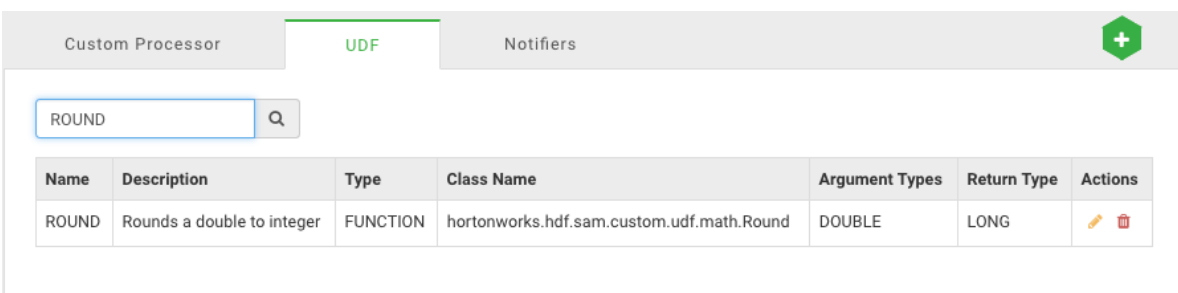
3. Adding UDFs (User Defined Functions) is easy to do within SAM. Follow the below steps to add Round UDF function to SAM.

   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:

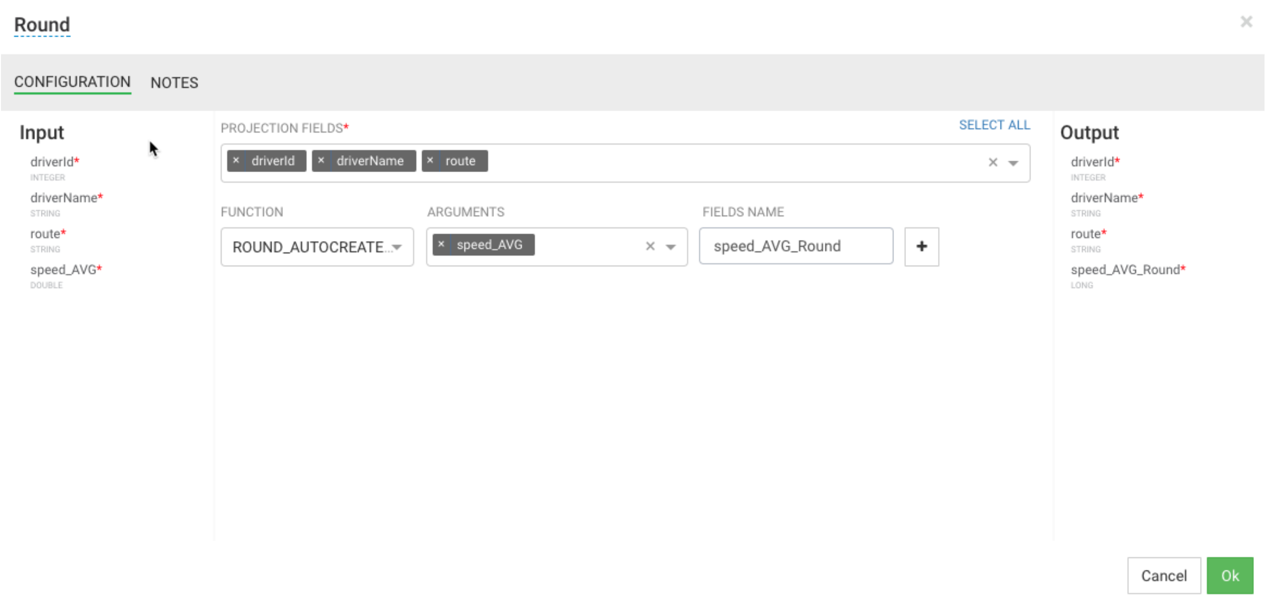


c. After uploading the UDF, you should see the new Round UDF created.



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:

Results

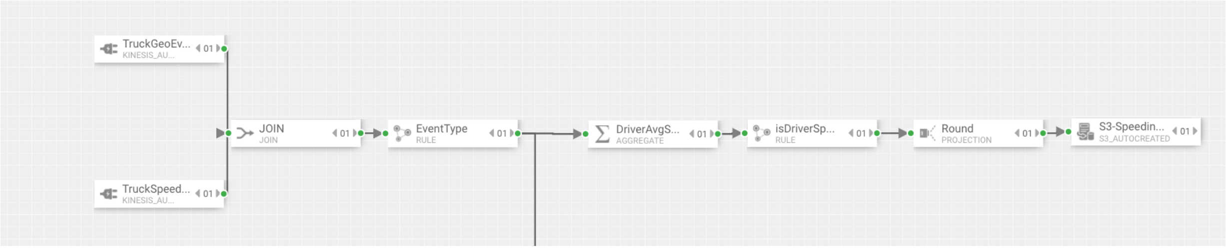


Streaming Alerts to an Analytics Engine for Dashboarding

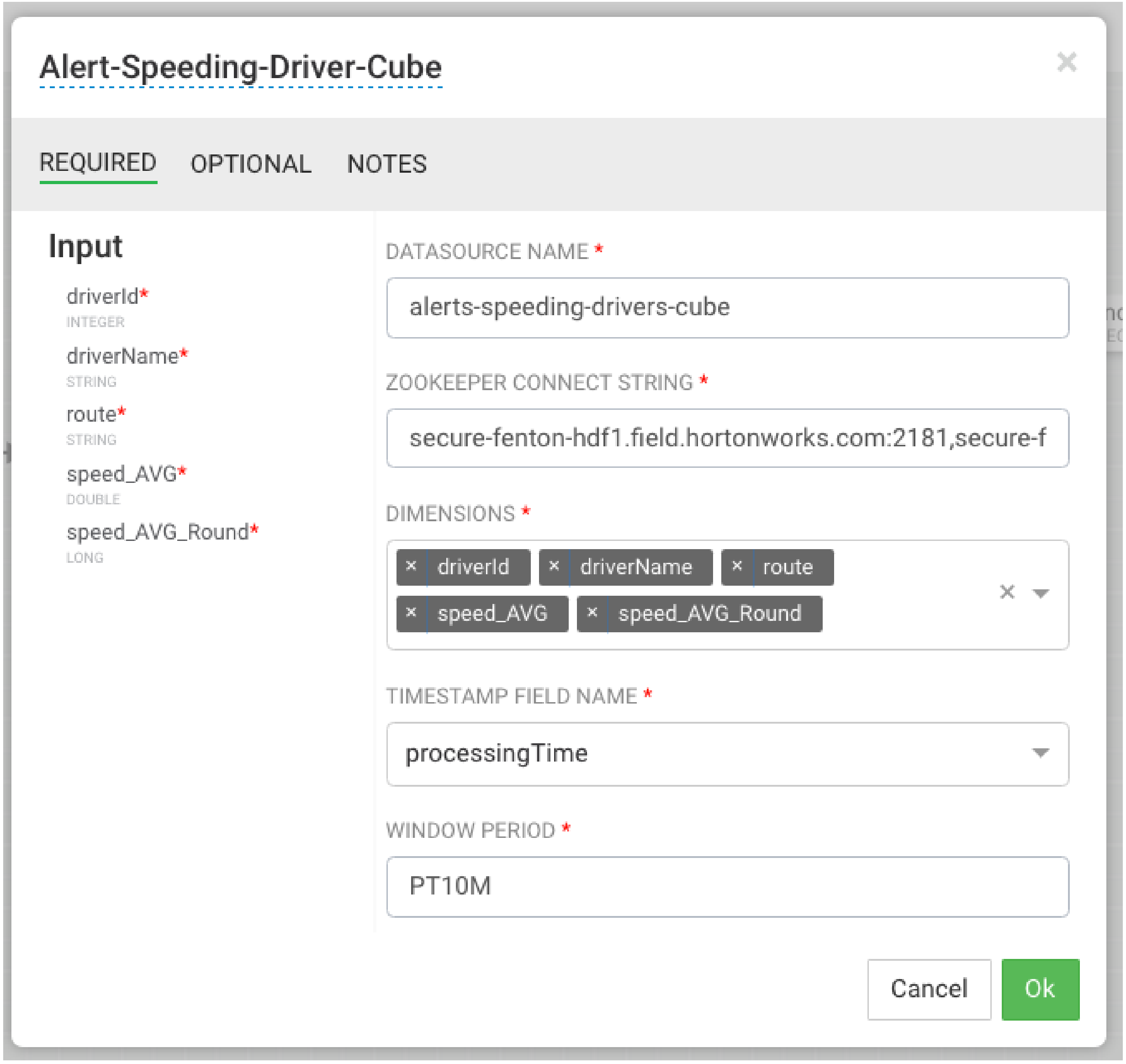
In addition to creating notification alerts, a common use case requirement is to send these alerts to a dashboard so they can be displayed and visualized. SAM offers this capability by allowing you to stream data into DRUID and then using Superset to create dashboards and visualizations.

Procedure

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:



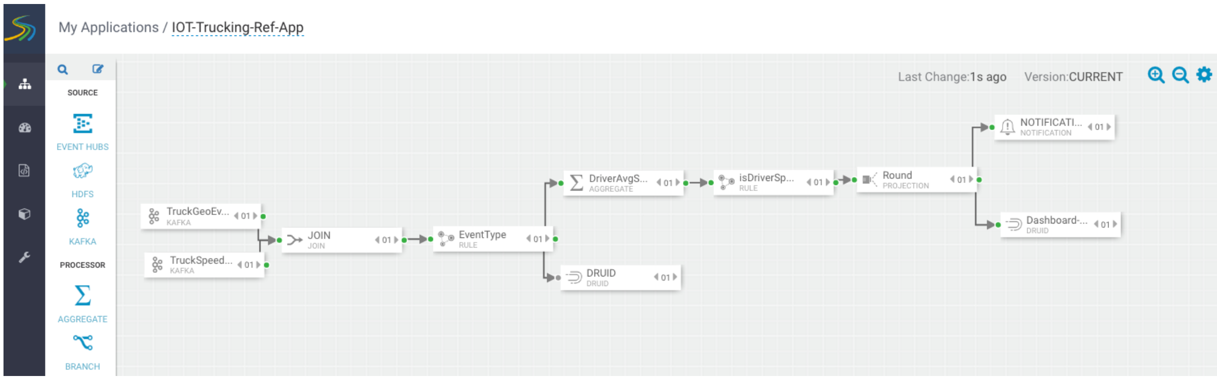
3. In the Creating Visualization Section, describe how to create dashboards for the **alerts-speeding-drivers-cube**.

Streaming Violation Events to an Analytics Engine for Descriptive Analytics

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.

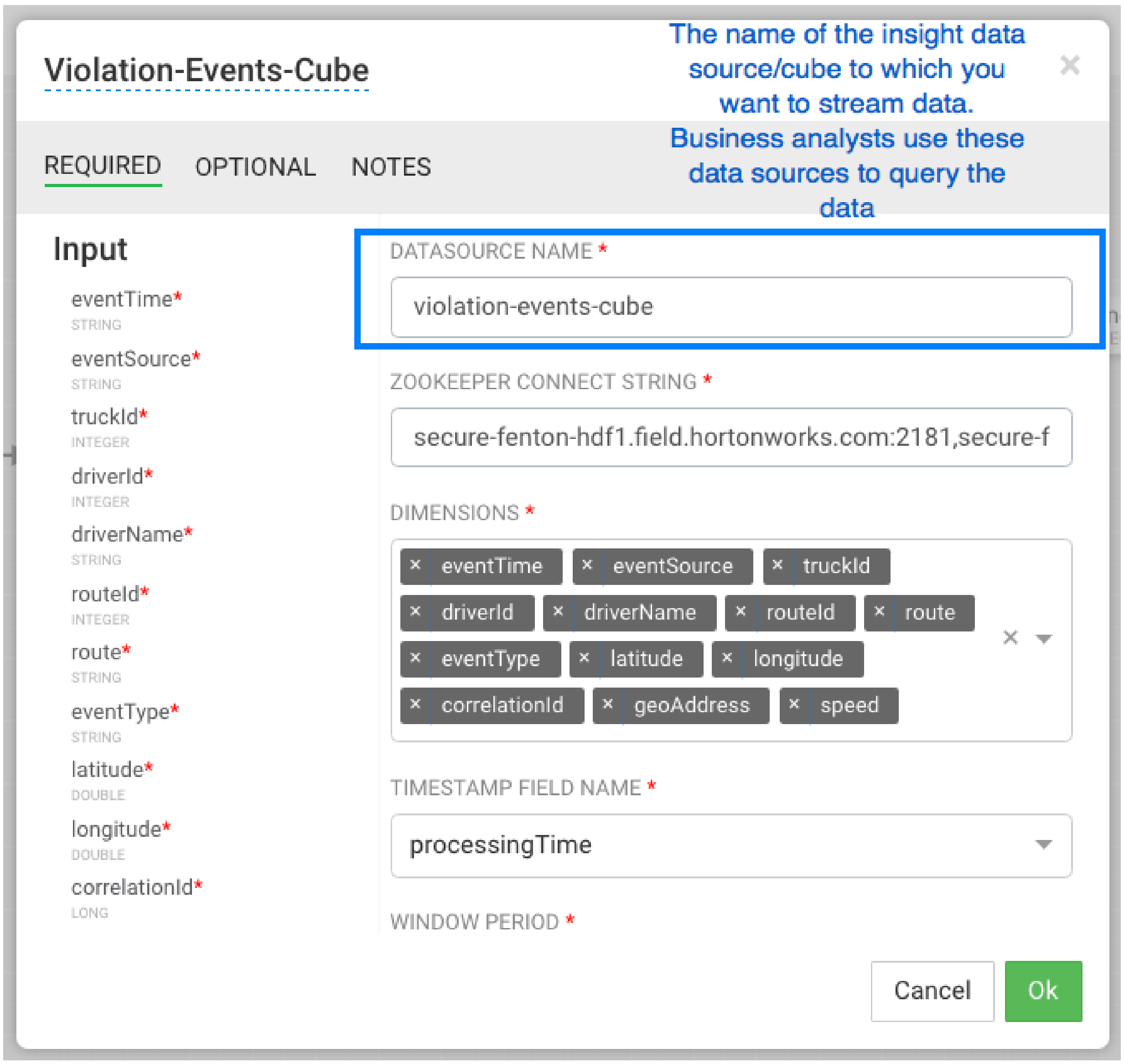
Procedure

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

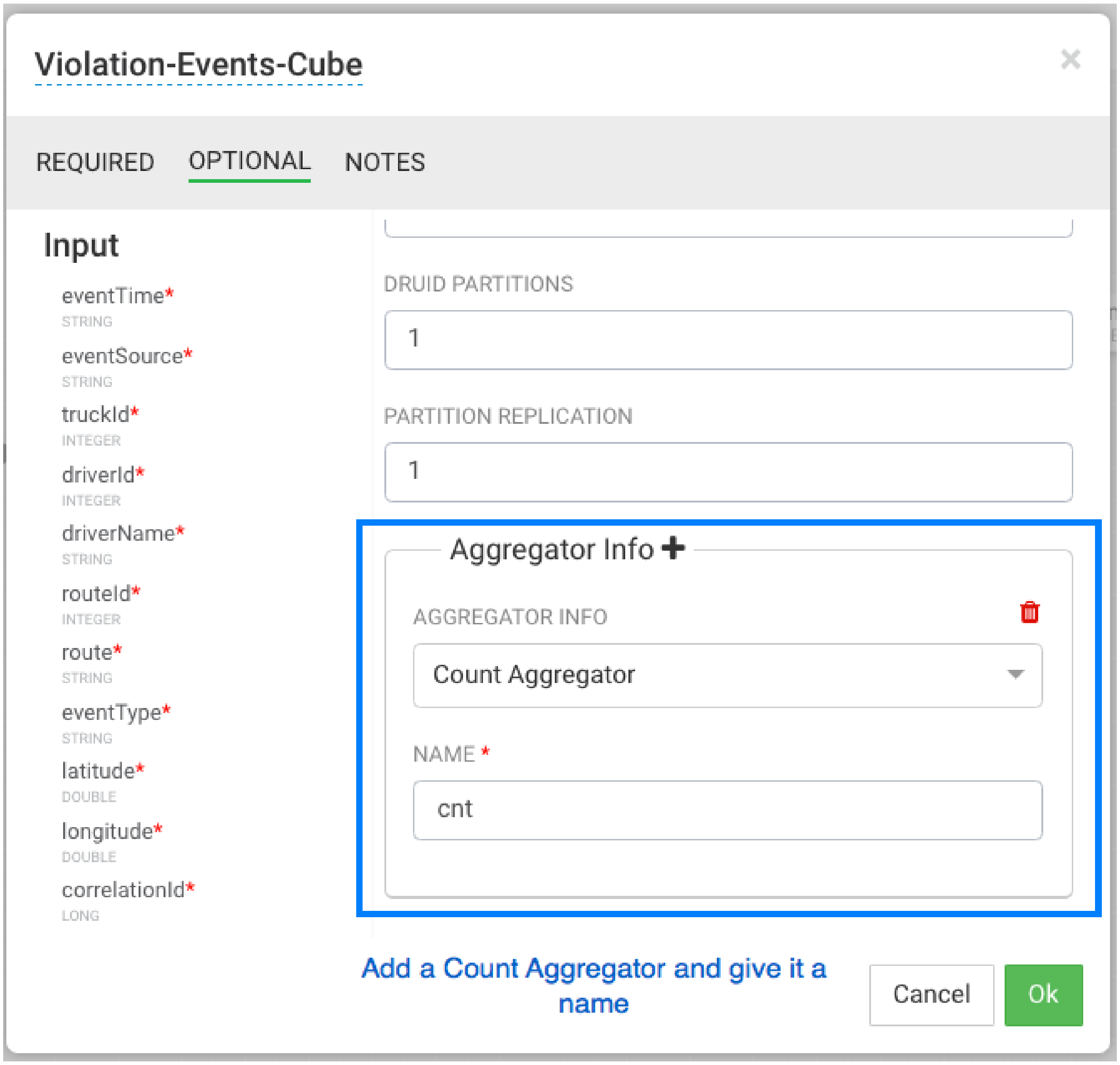


2. Configure the Druid processor.

   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

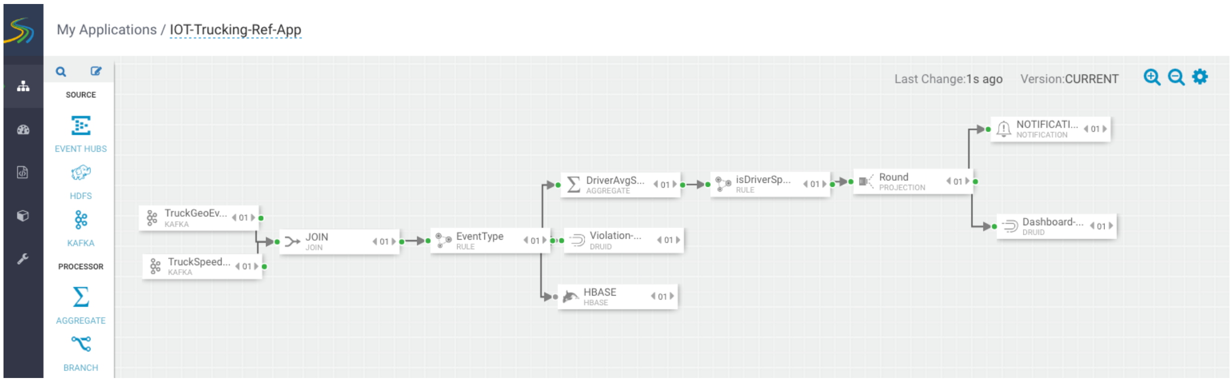


Streaming Violation Events into a Data Lake and Operational Data Store

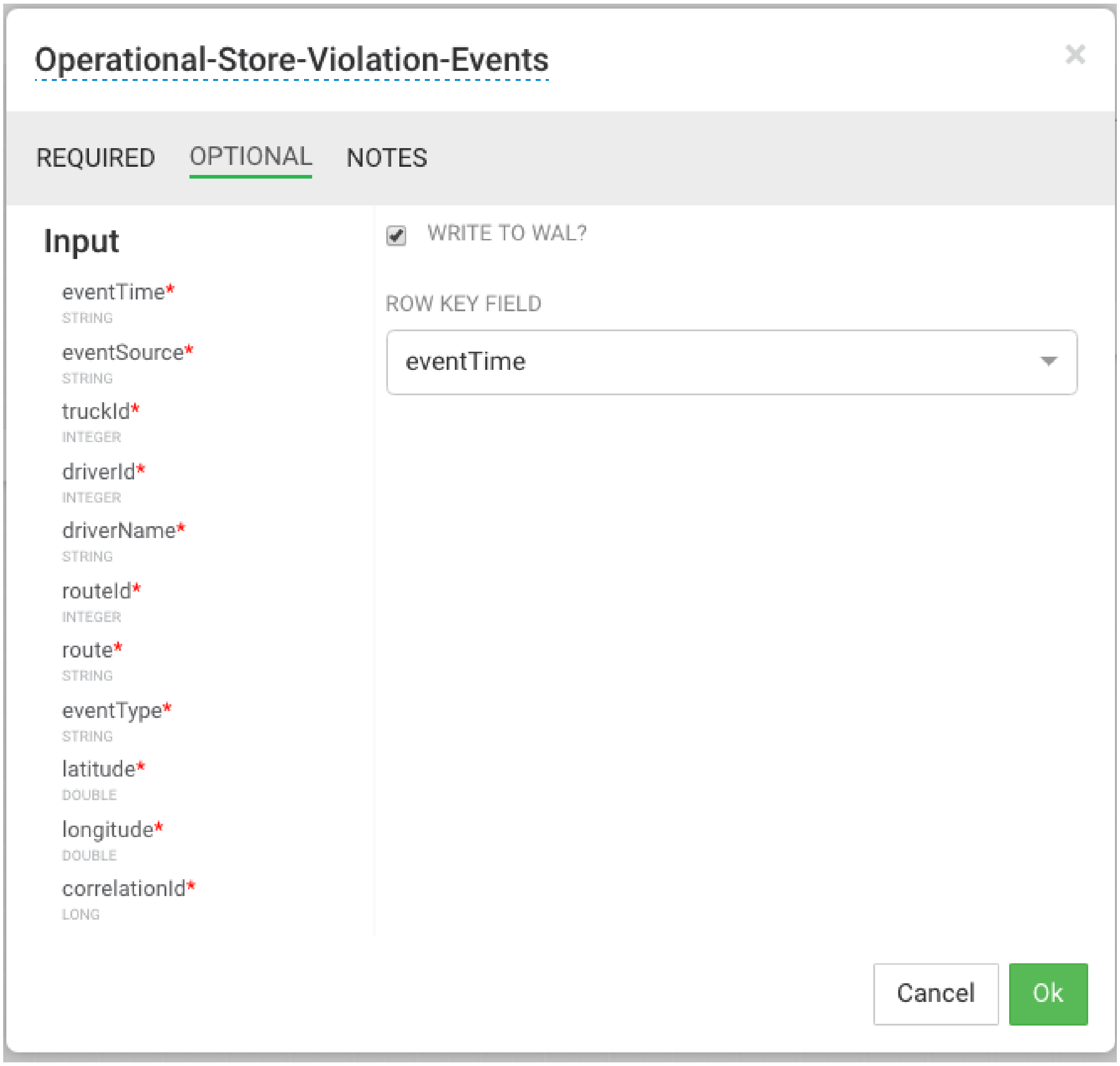
Another common requirement is to stream data into an operational data store like HBase to power real-time web applications as well as a data lake powered by HDFS for long term storage and batch ETL and analytics.

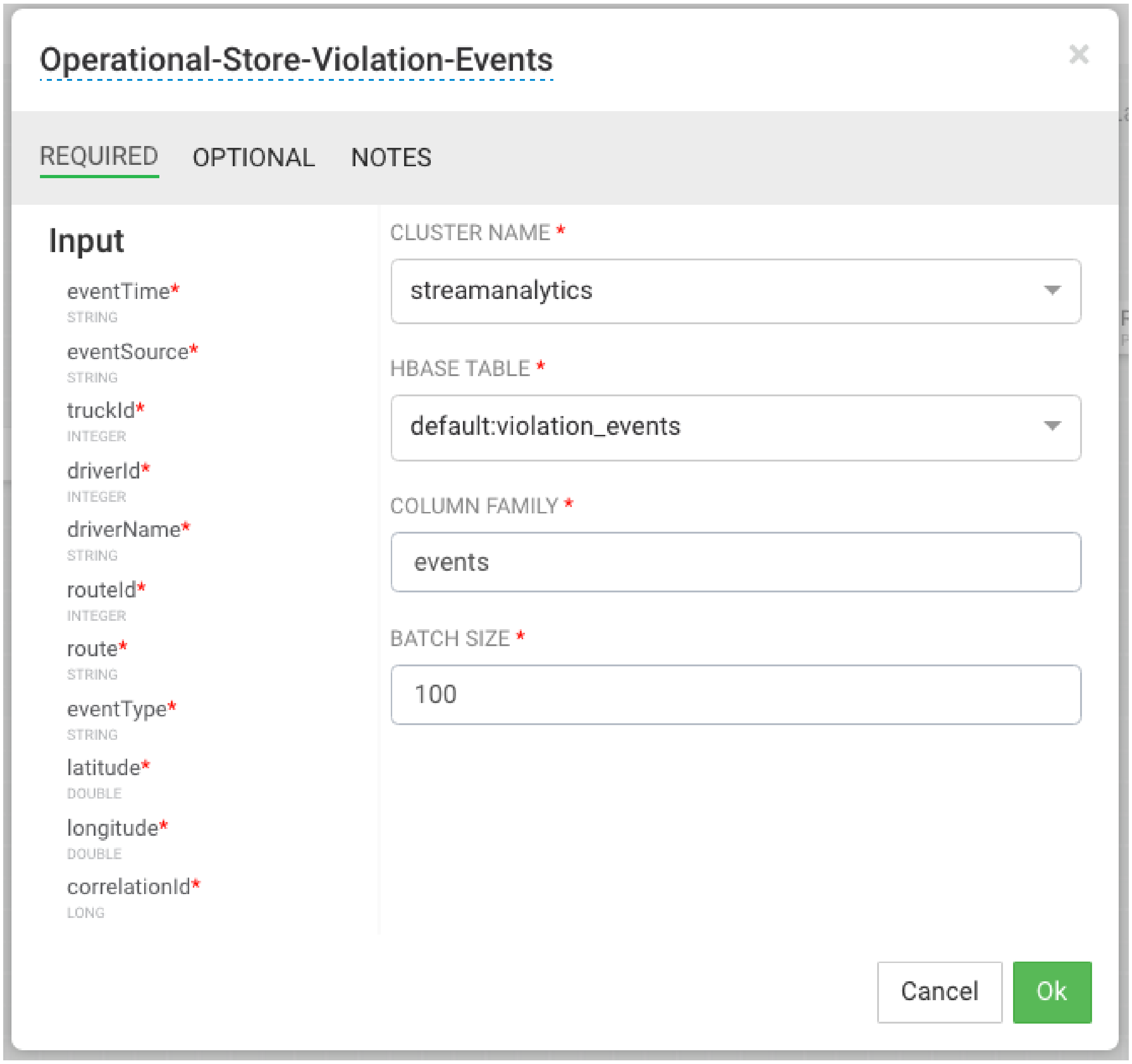
Procedure

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

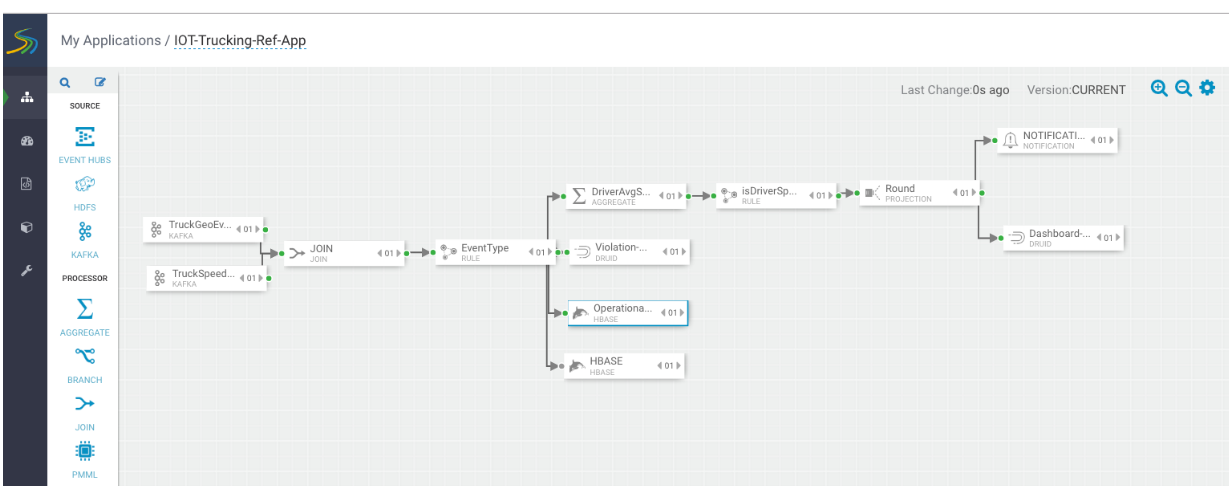


2. Configure the Hbase Sink using the following parameters.



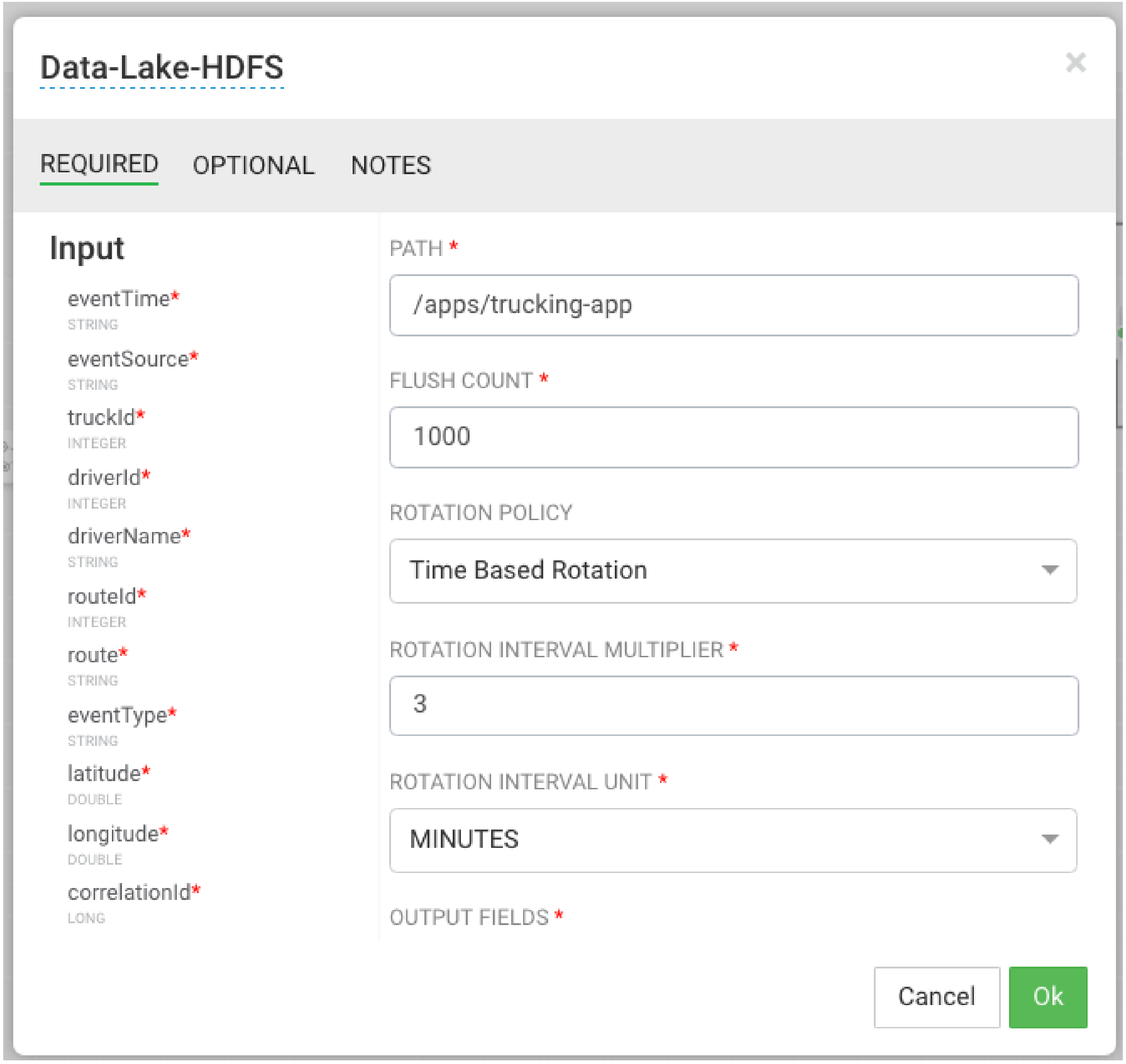


3. Drag the HDFS sink to the canvas and connect it to the ViolationEvents Rule processor.



4. Configure HDFS as below.

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



Deploy a SAM Application

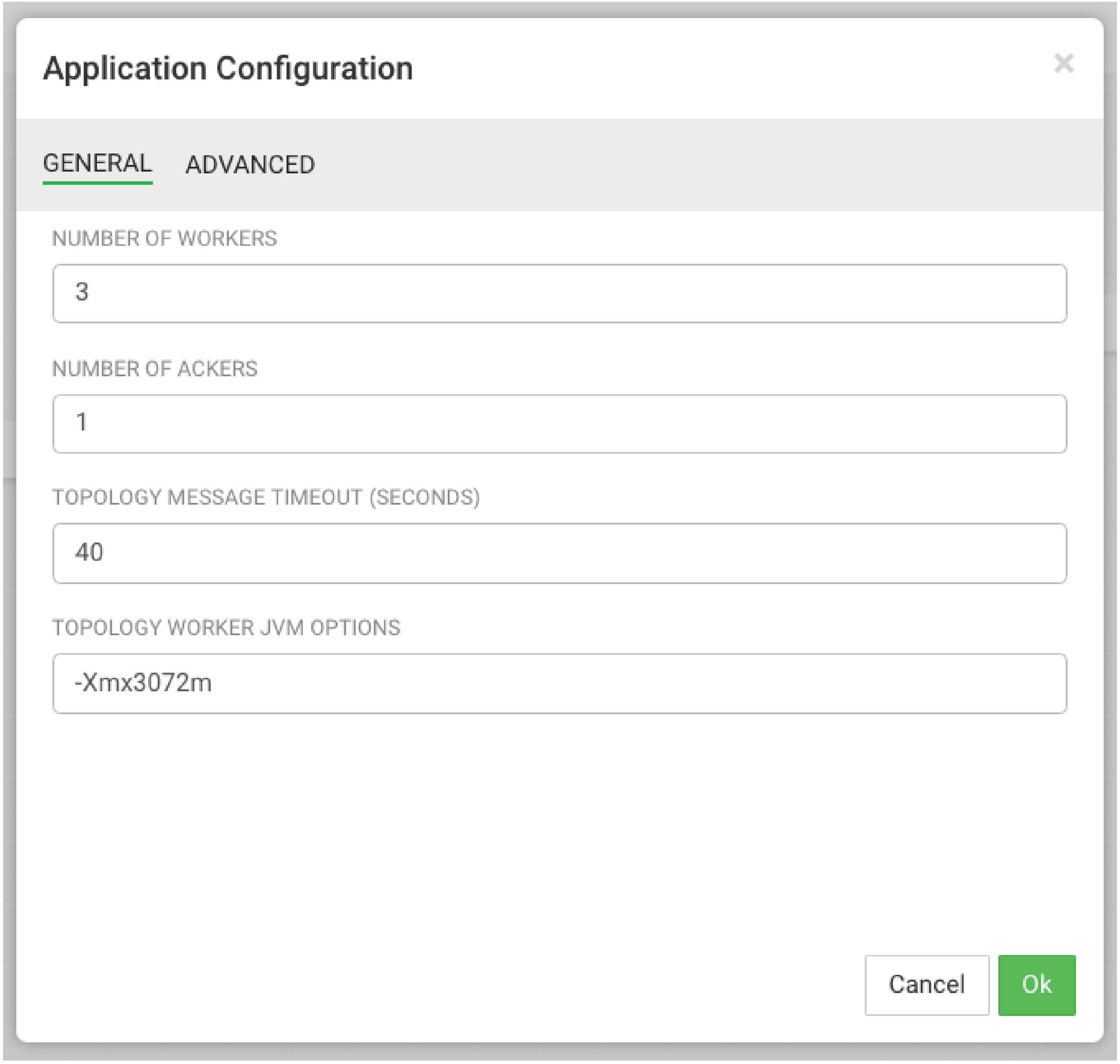
Configure Deployment Settings

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.

Procedure

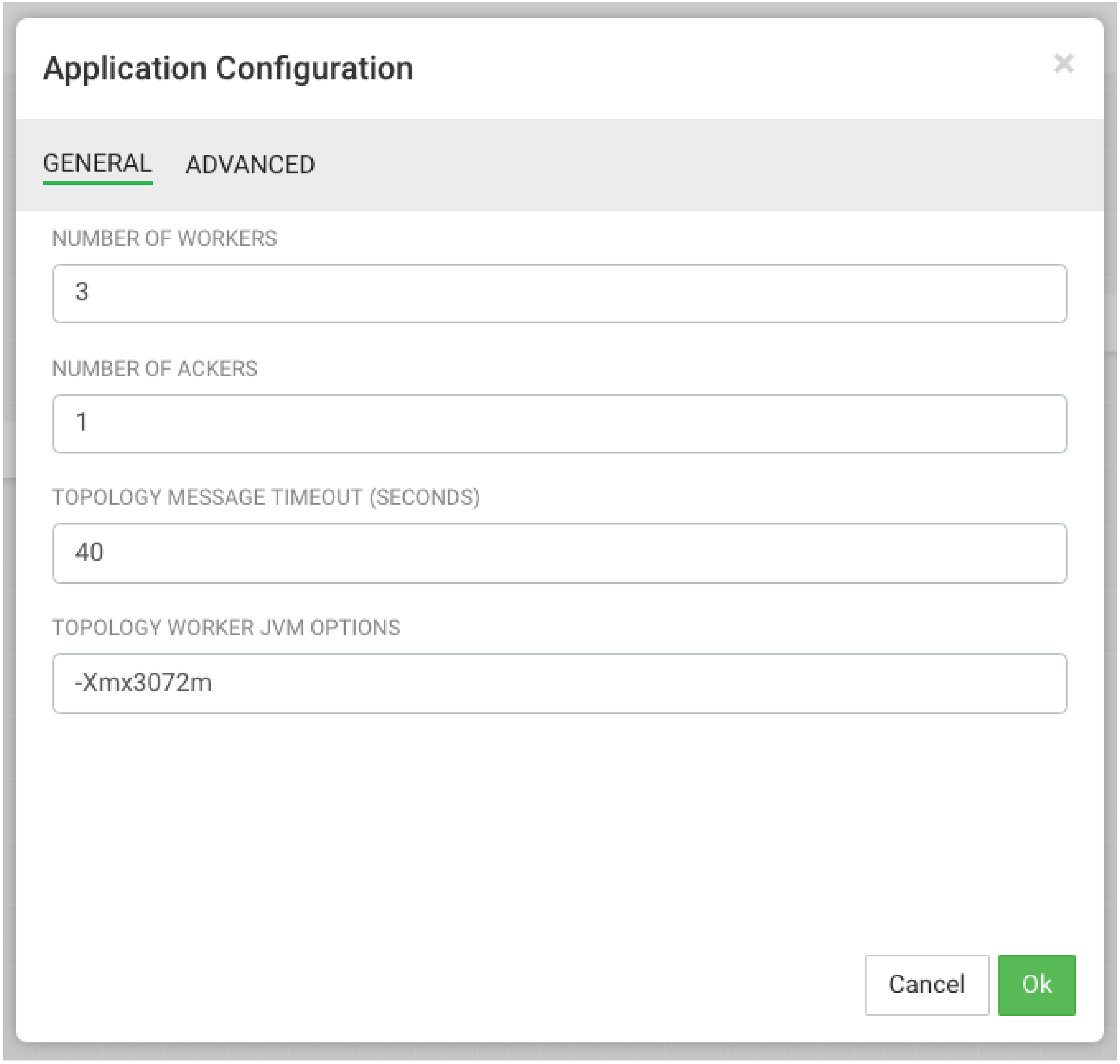
1. Click the gear icon at the top right corner of the canvas to display the **Application Configuration** dialog.
2. Increase **Number of Workers** to 5.
3. Set **Topology Worker JVM Options** to -Xmx3072m.

Example



Deploy the App

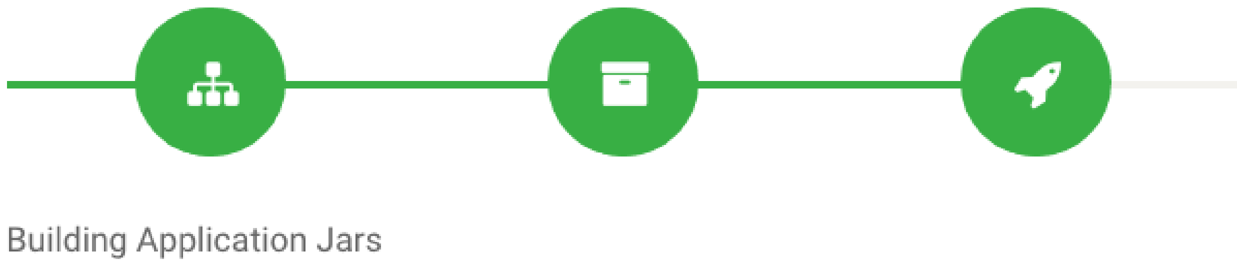
After you have configure the application's deployment settings, click the **Deploy** button at the lower right of the canvas.



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

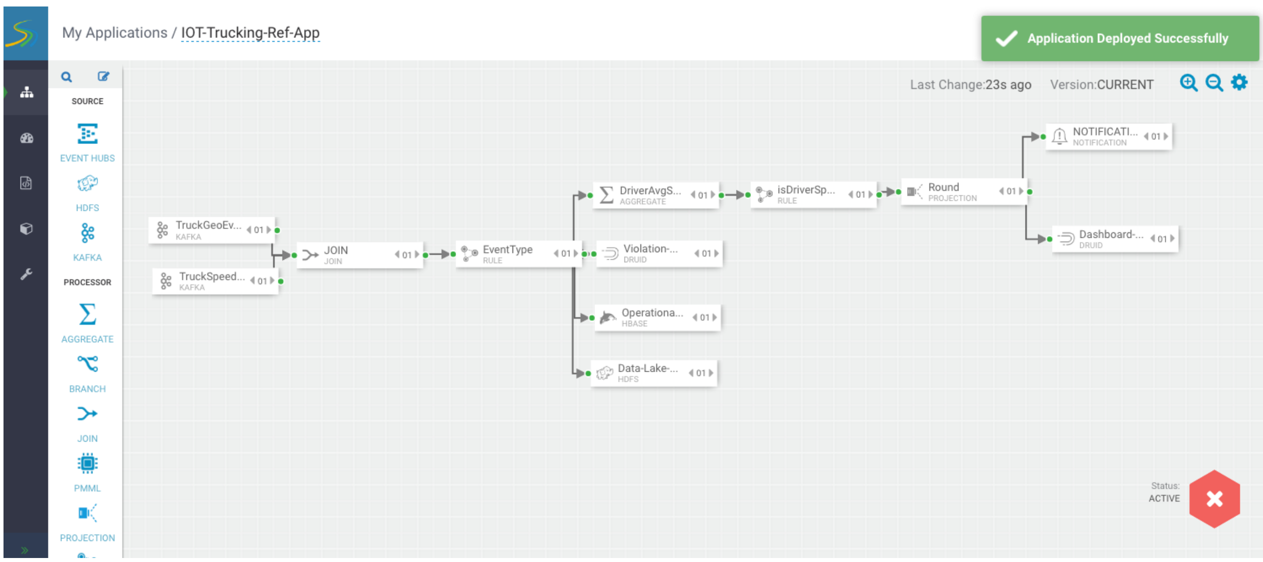
1. Construct the configurations for the different big data services used in the stream app.
2. Create a deployable jar of the streaming app.
3. Upload and deploy the app jar to streaming engine server.

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.

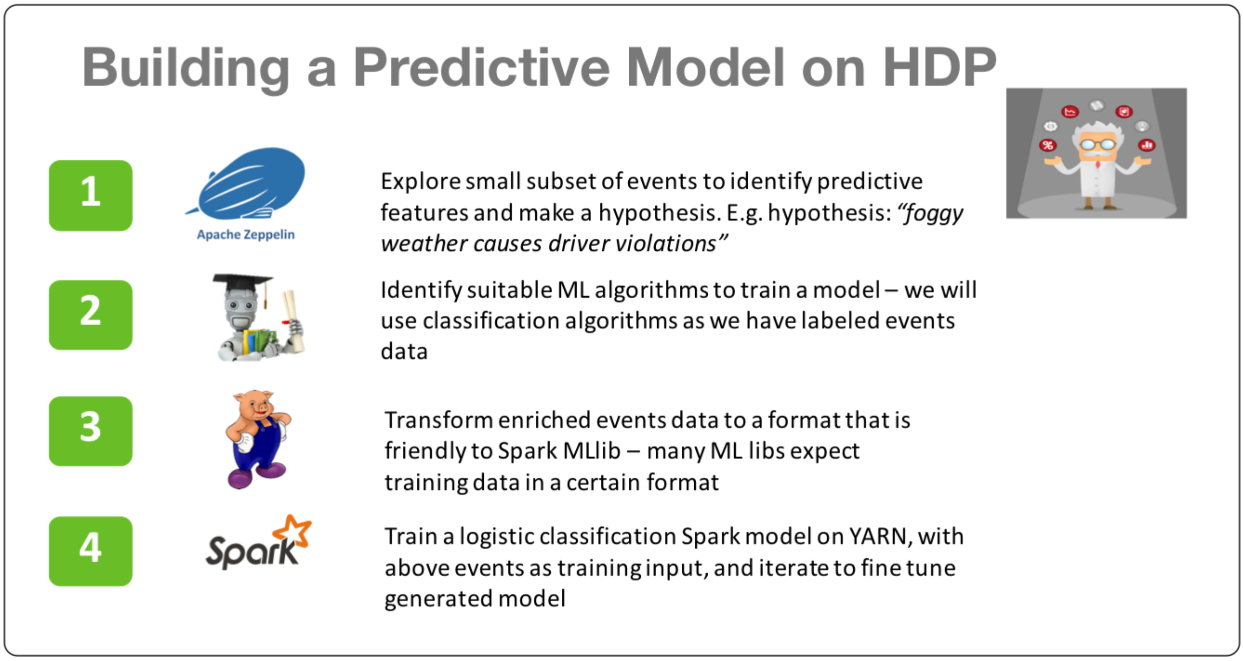


Advanced: Performing Predictive Analytics on the Stream using SAM

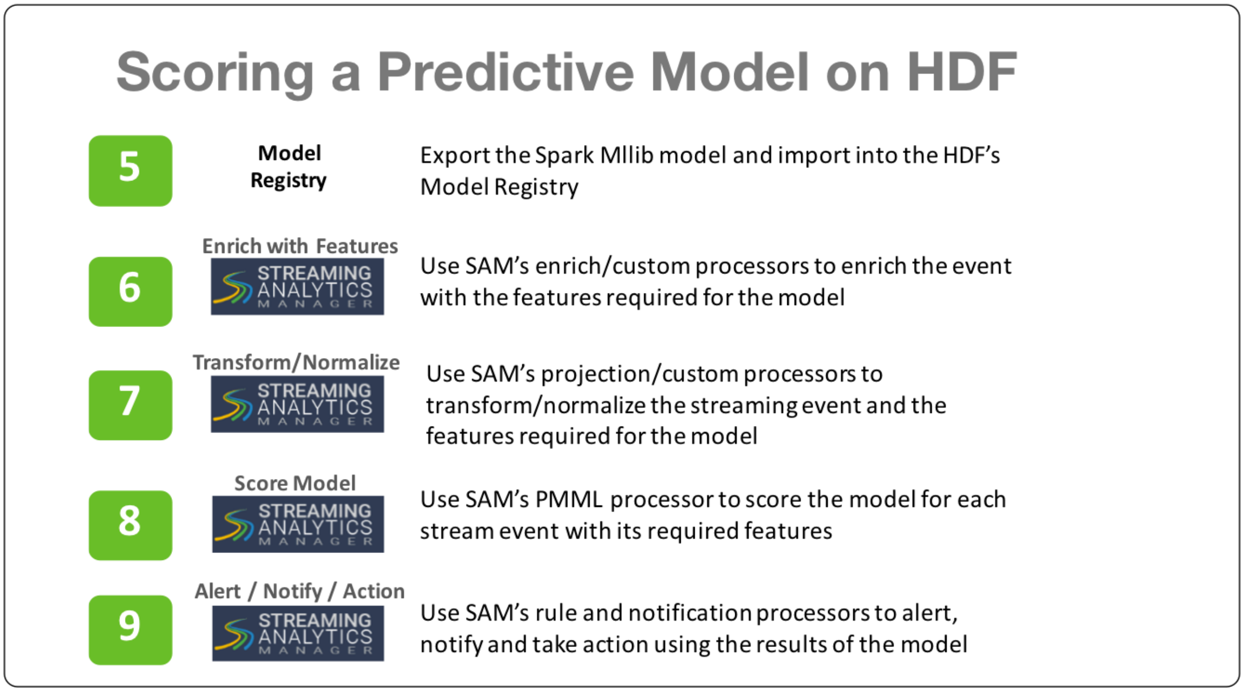
Requirement 10 of this use case states the following:

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.



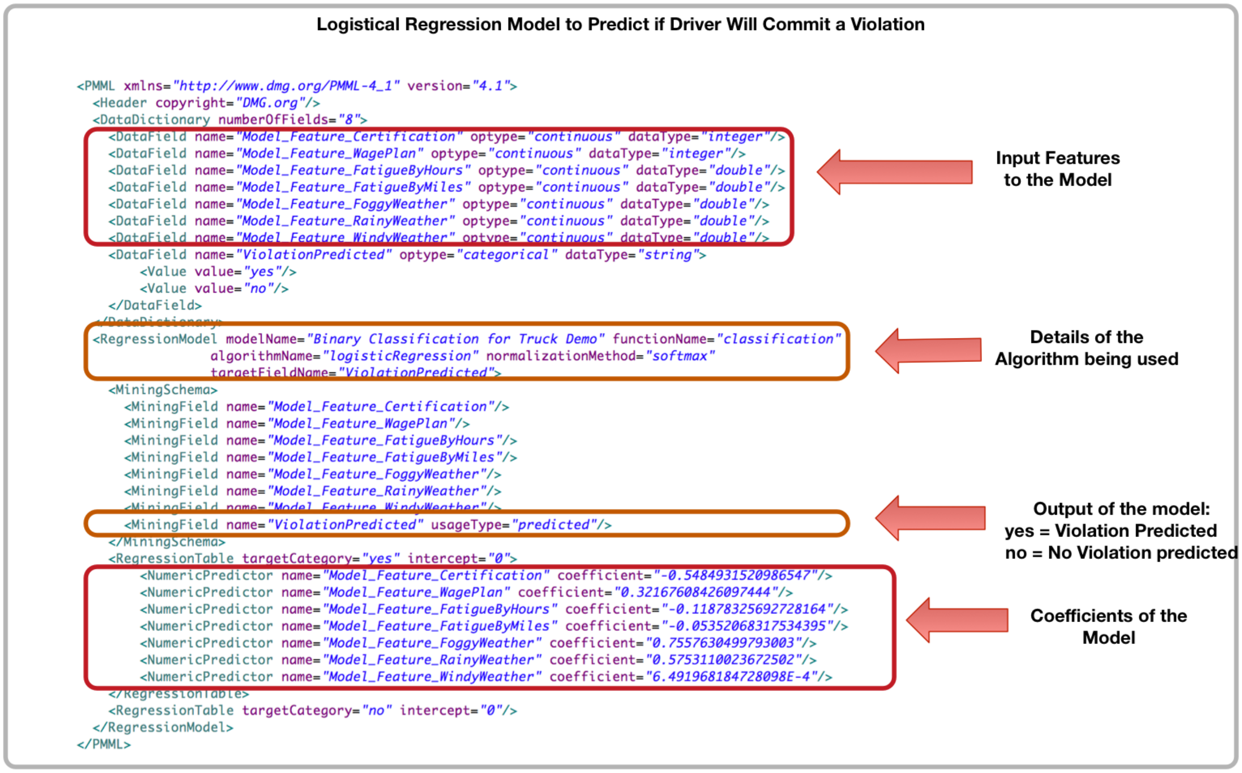
Once the model has been trained and optimized, you can create insights by scoring the model in real-time as events are coming in. The next set of steps in the life cycle score the model in real-time using HDF components.



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

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.

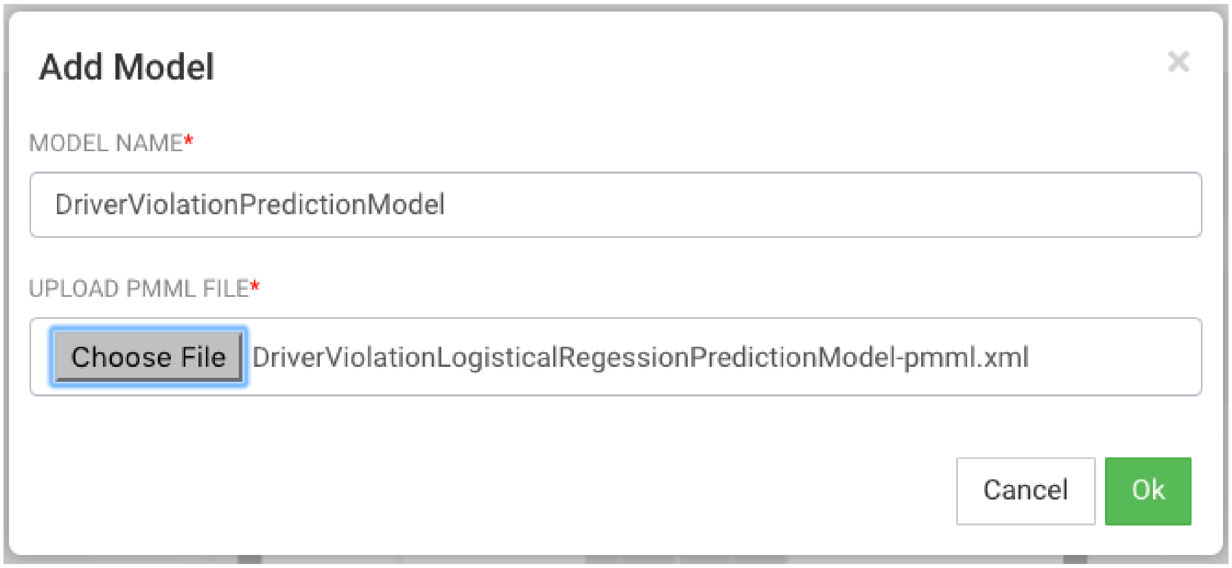


Export the Model into SAM's Model Registry

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

Procedure

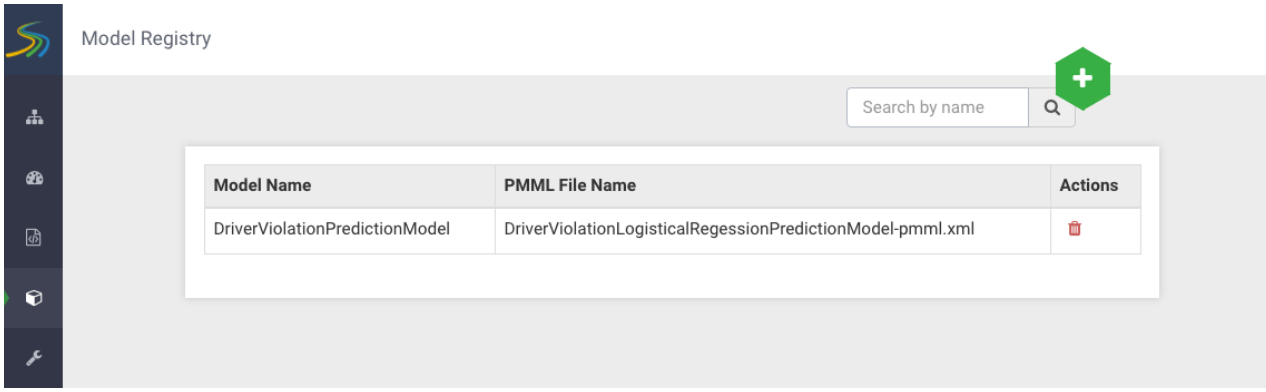
1. Download this PMML model and save it locally with an .xml extension.
2. Select the **Model Registry** menu item.
3. Click the **+** icon.
4. Give your PMML model a name.
5. From **Upload PMML File**, select the PMML file you just downloaded.



6. Click **Ok**.

Results

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



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

Upload Custom Processors and UDFs for Enrichment and Normalization

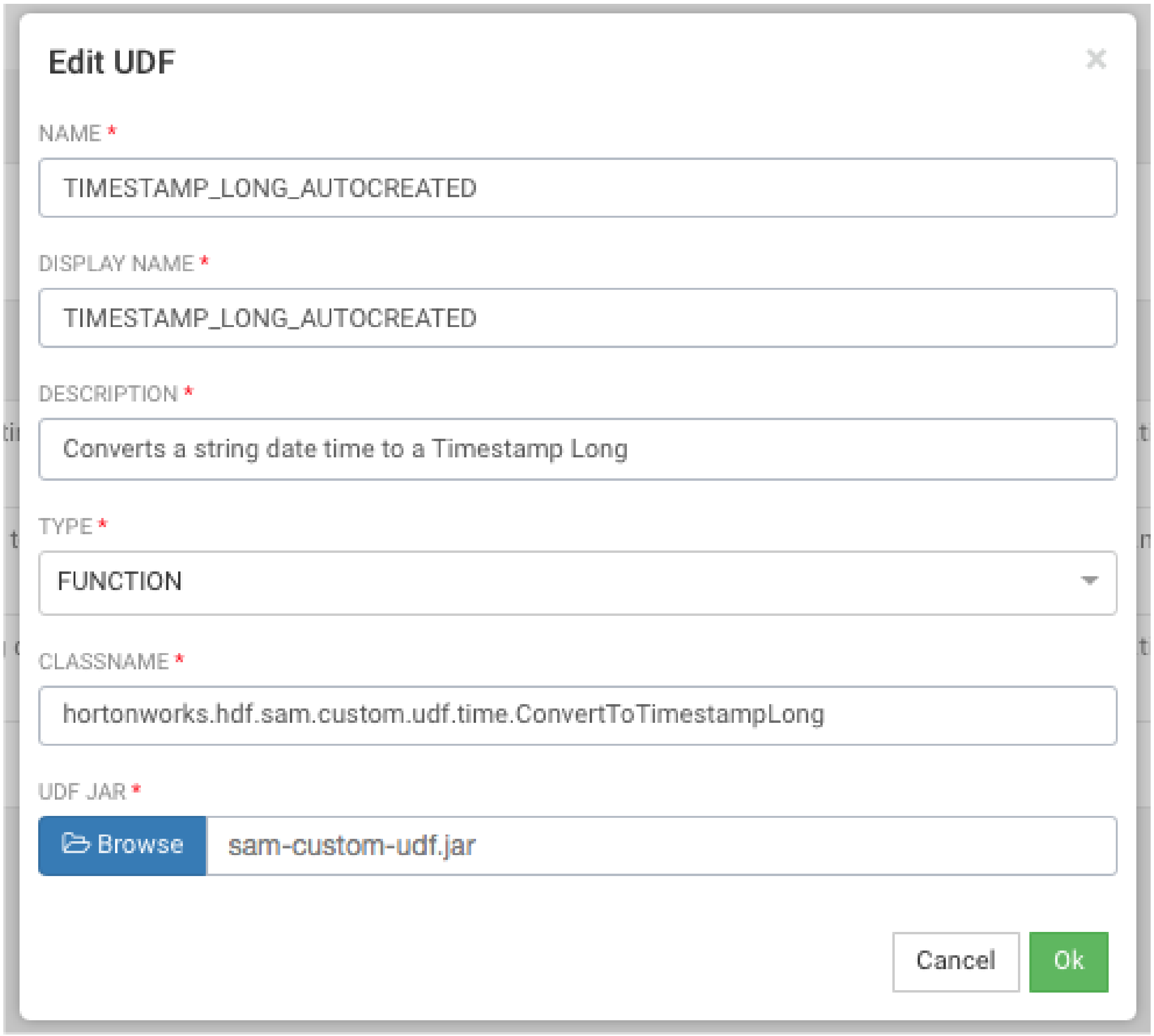
To perform the above enrichment and normalization, upload the custom UDFs and processors you downloaded in the previous section.

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.

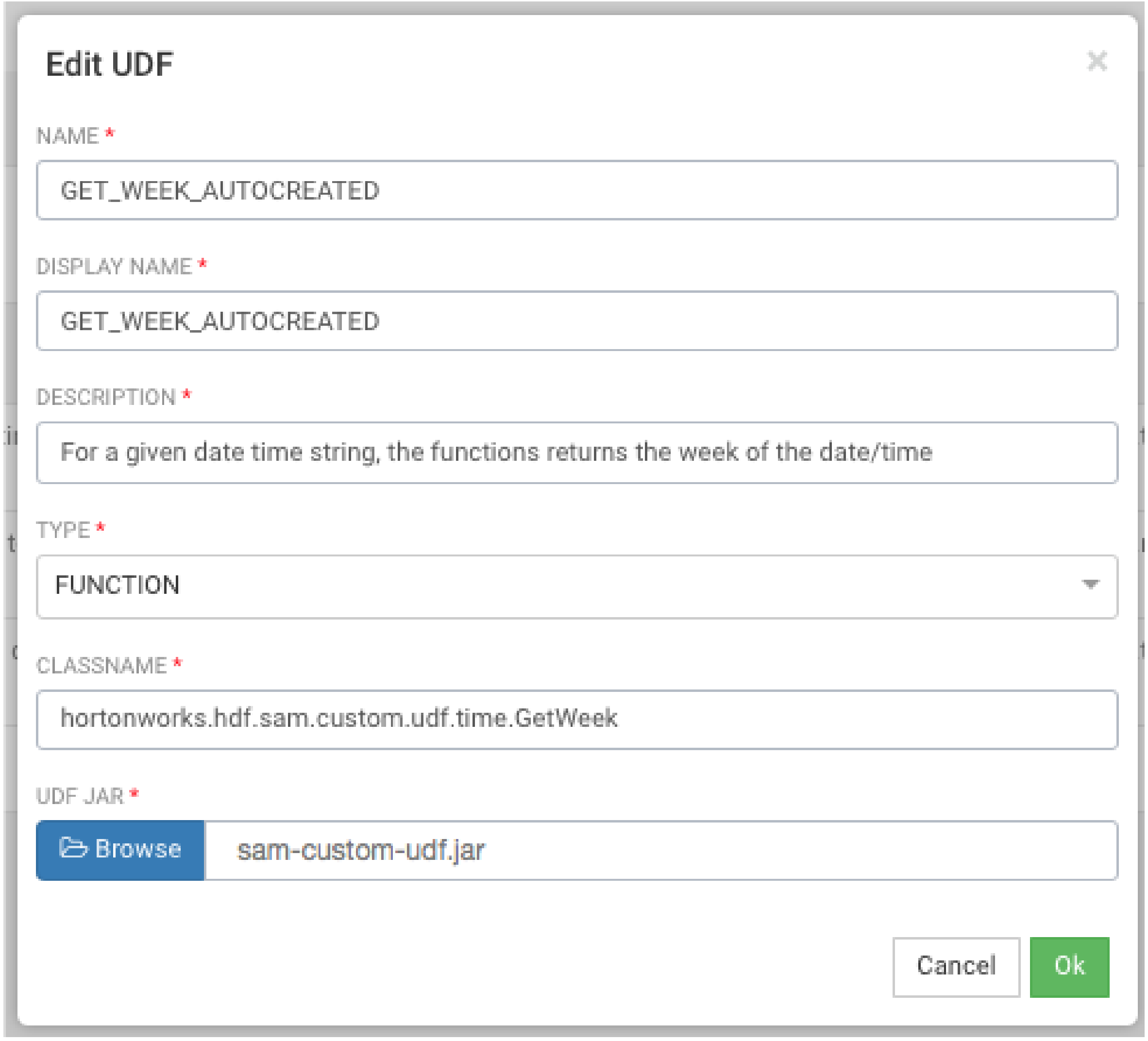
   The jar for this UDF is located in SAM_EXTENSIONS/custom-udf.
3. Configure the UDF with the following values:



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:

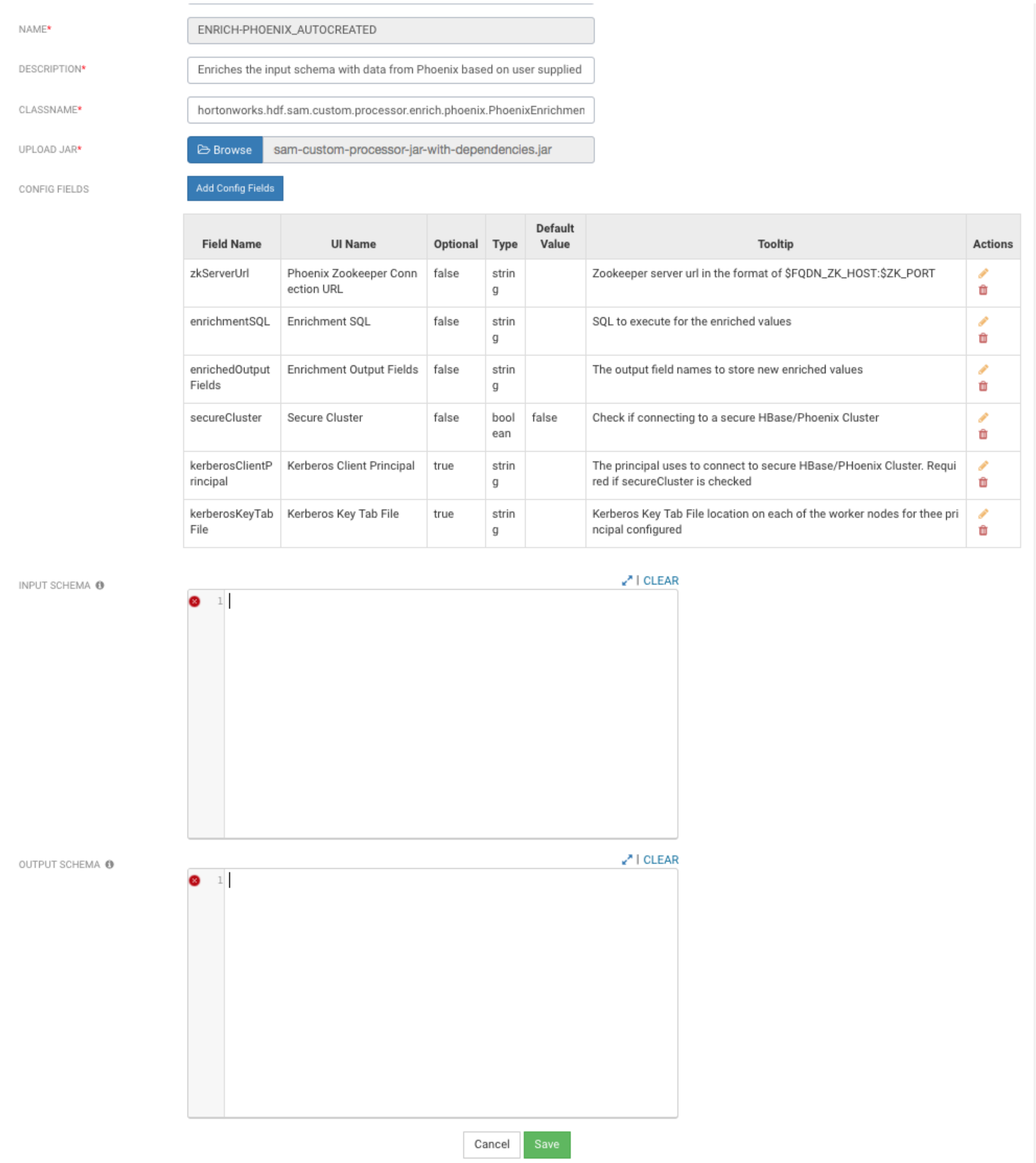


Upload Custom Processors

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.PhoenixEnrichmentSecureProcessor
- **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:

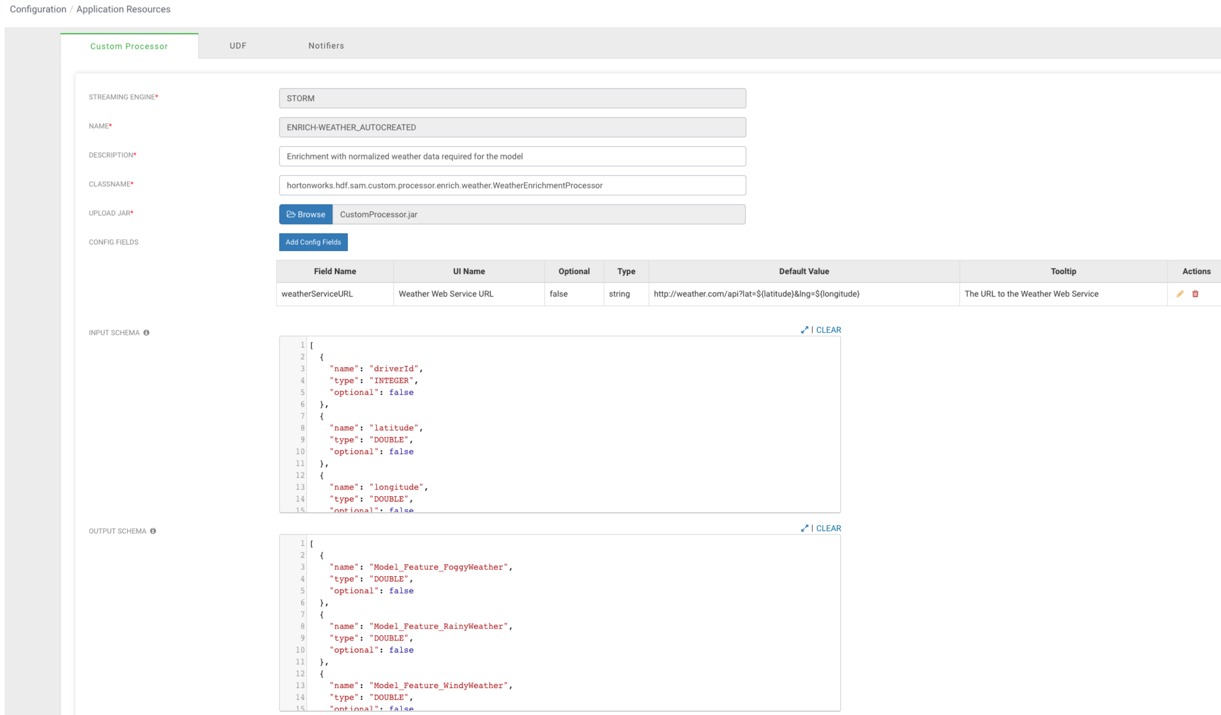
a. **Field Name** – kerberosClientPrincipal
b. **UI Name** – Kerberos Client Principal
c. **Optional –** true
d. **Type** – string
e. **ToolTip –** The principal uses to connect to secure HBase/Phoenix Cluster. Required if secureCluster is checked.

- 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:

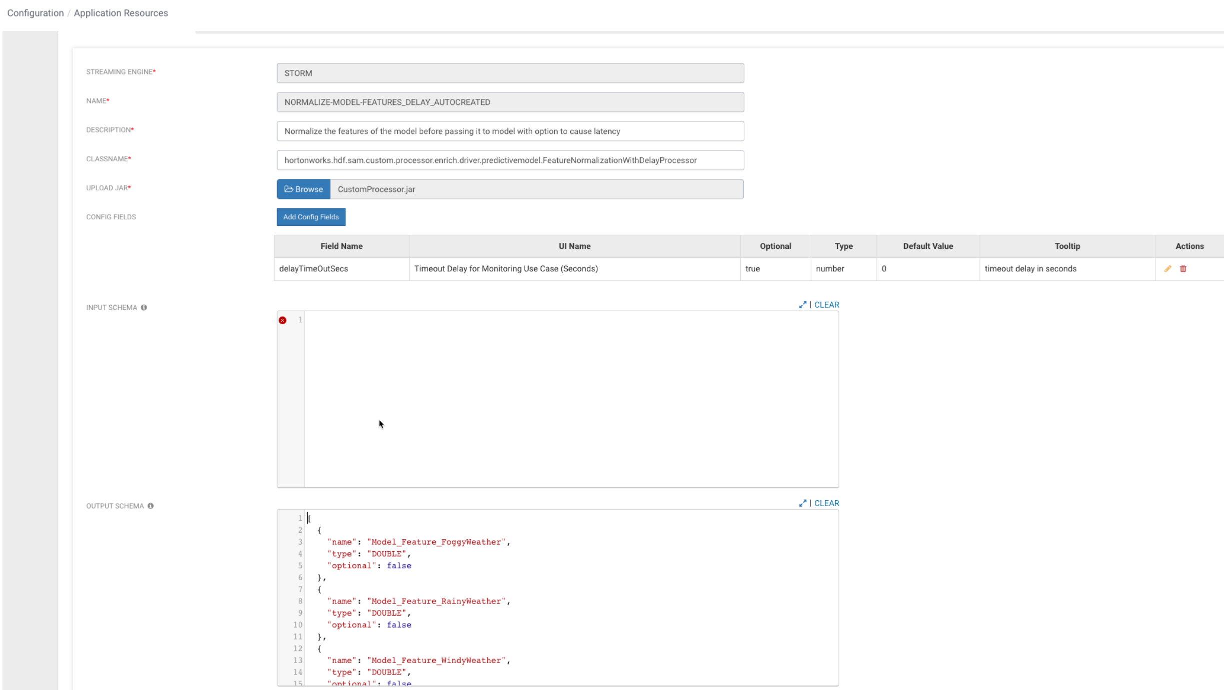
- **Field Name** – weatherServiceURL
- **UI Name** – Weather Web Service URL
- **Optional** – false
- **Type** – string
- **Tooltip** – The URL to the Weather Web Service

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

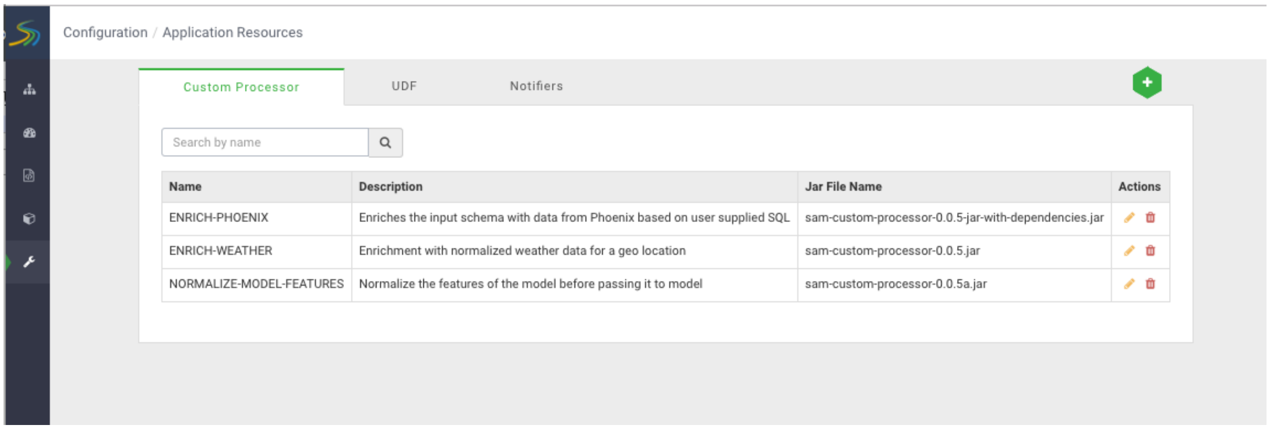
- **Streaming Engine –** Storm
- **Name** – NORMALIZE-MODEL-FEATURES
- **Description –** Normalize the features of the model before passing it to model
- **ClassName –** hortonworks.hdf.sam.custom.processor.enrich.driver.predictivemodel.FeatureNormalizationProcessor
- **Upload Jar** – The jar for this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

Input and Output Schema for NORMALIZE-MODEL-FEATURES

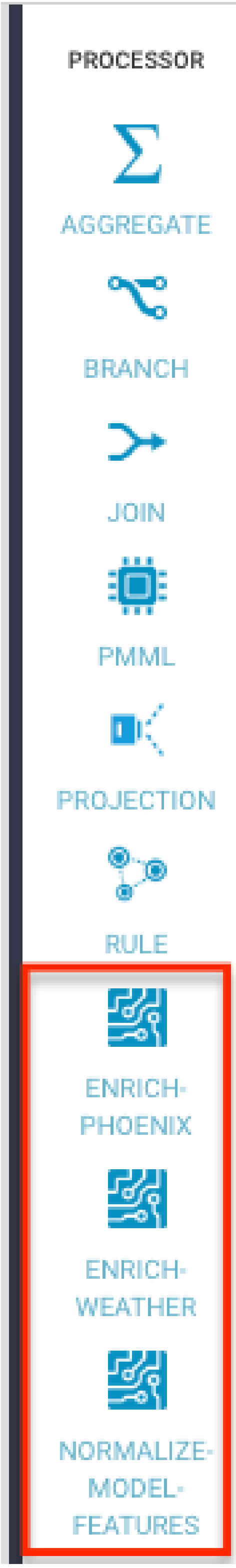
- Copy this output schema and paste into the **OUTPUT SCHEMA** text area box

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.

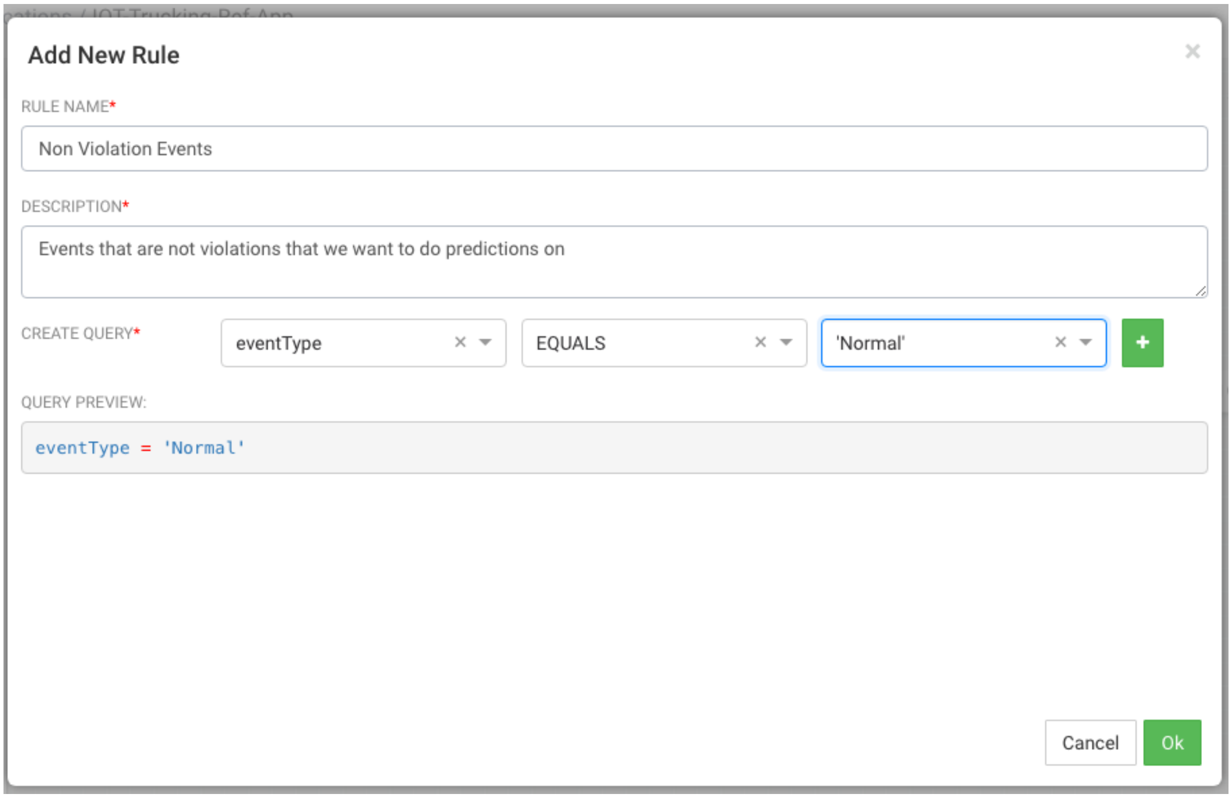


Scoring the Model in the Stream using a Streaming Split Join Pattern

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.

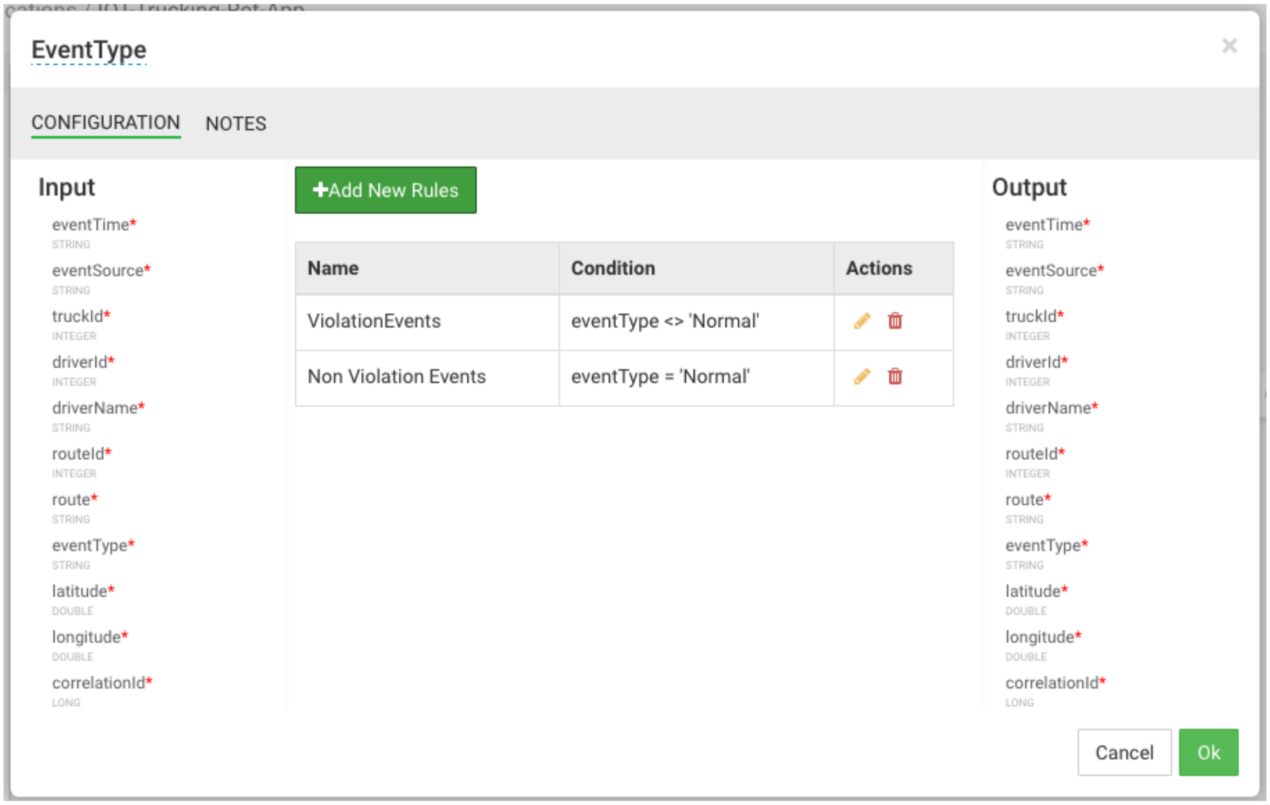
Procedure

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:



Results

Your new rule is added to the Event Type processor.



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.

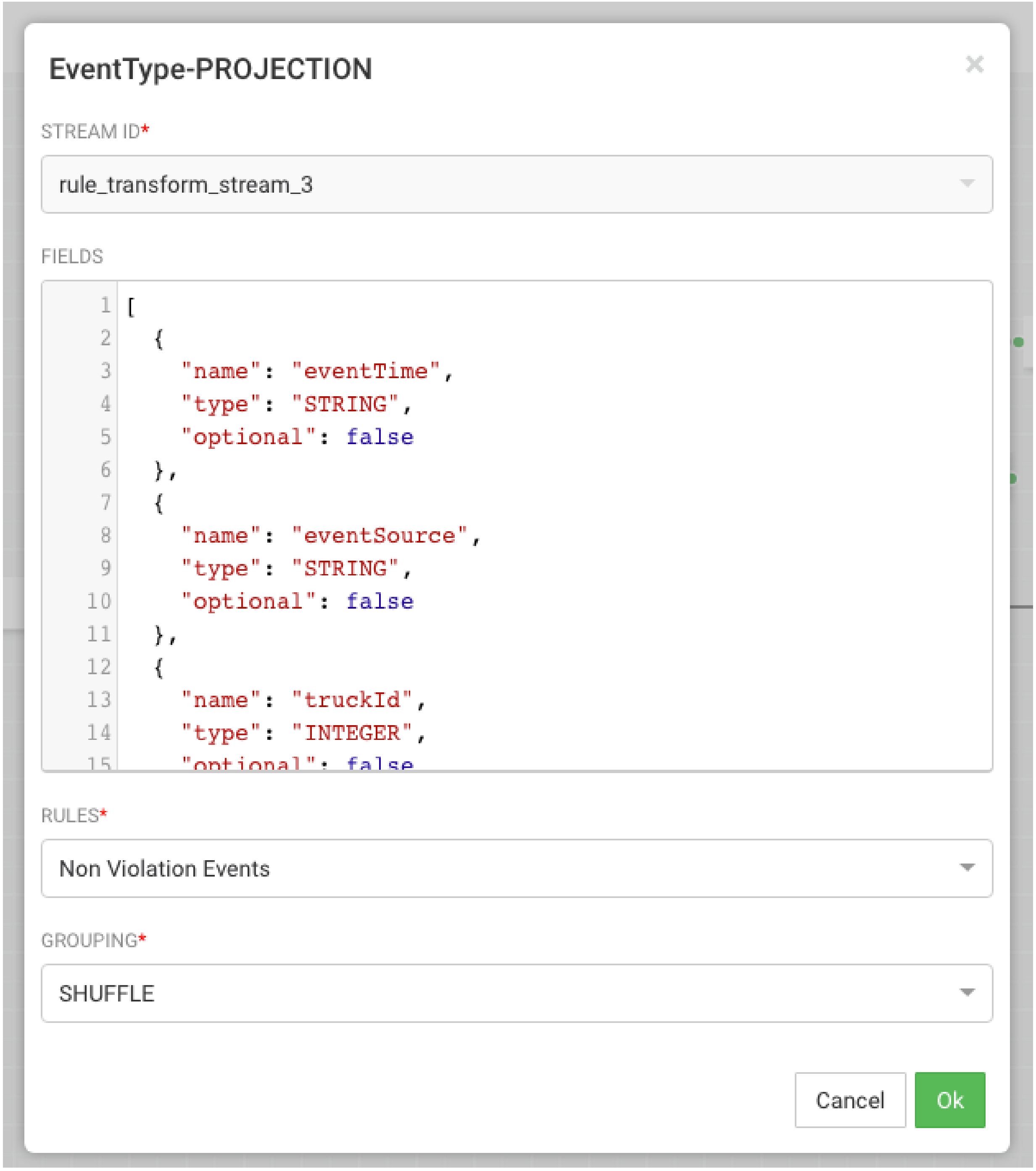
To do this, use the split join pattern to split the stream into three, perform the enrichment in parallel, and then re-join the three streams.

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.

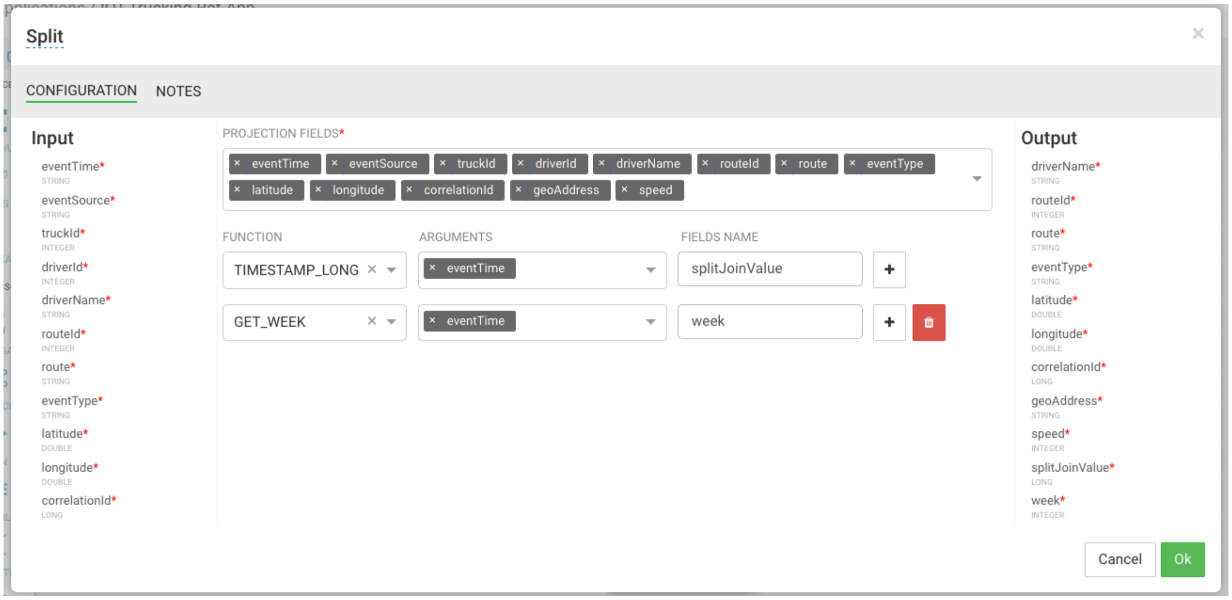
   To do this, drag the projection processor to the canvas and create a connection from the EventType rule processor to this projection processor.

   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:



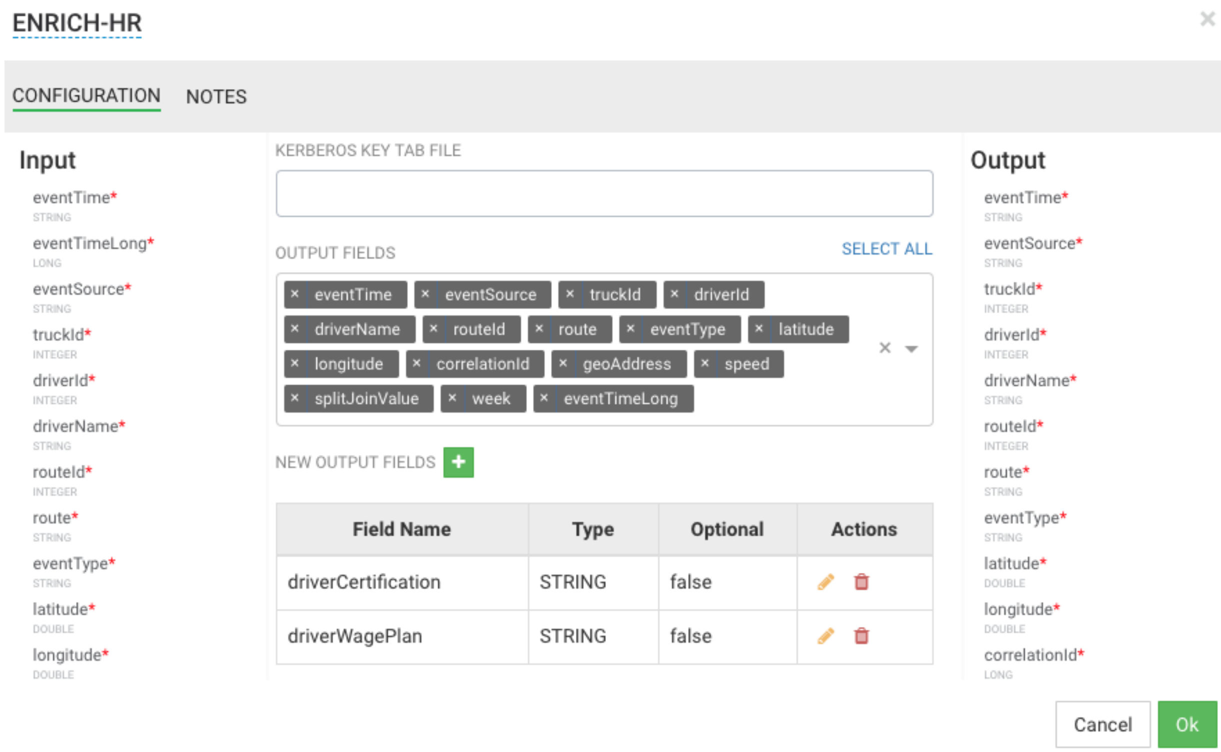
Steps for Splitting the Stream into Three to Perform Enrichments in Parallel

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

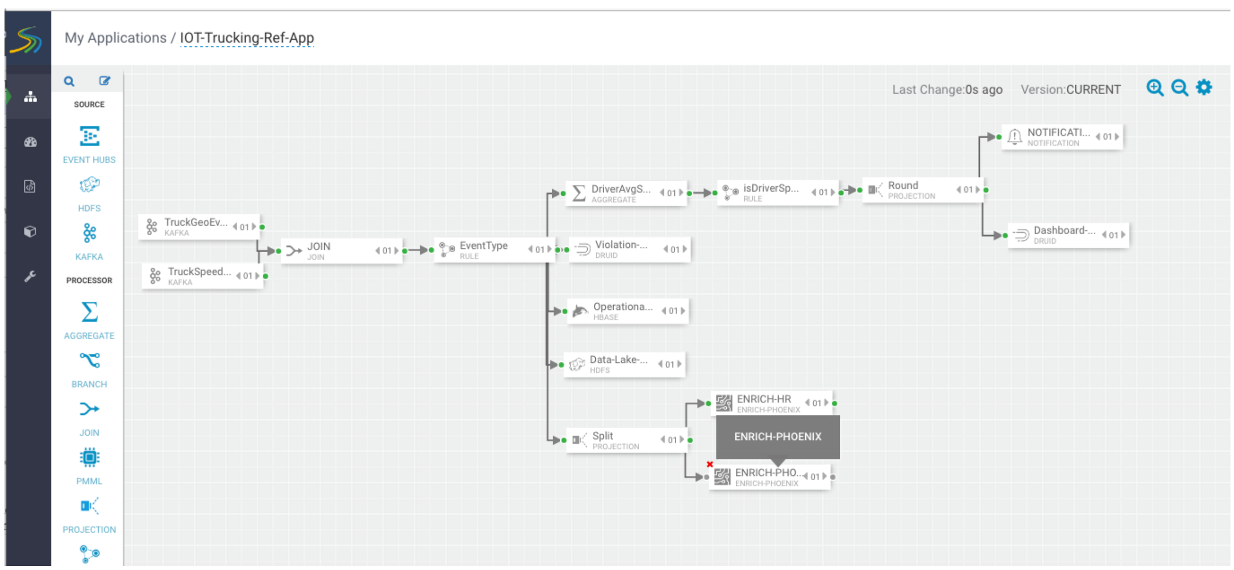
   To do the first split to enrichment the wage and certification status of driver, drag the "ENRICH-PHOENIX" processor to the canvas and connect it from the Split project processor.
2. Configure the enrich processor with the following parameters:

   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:

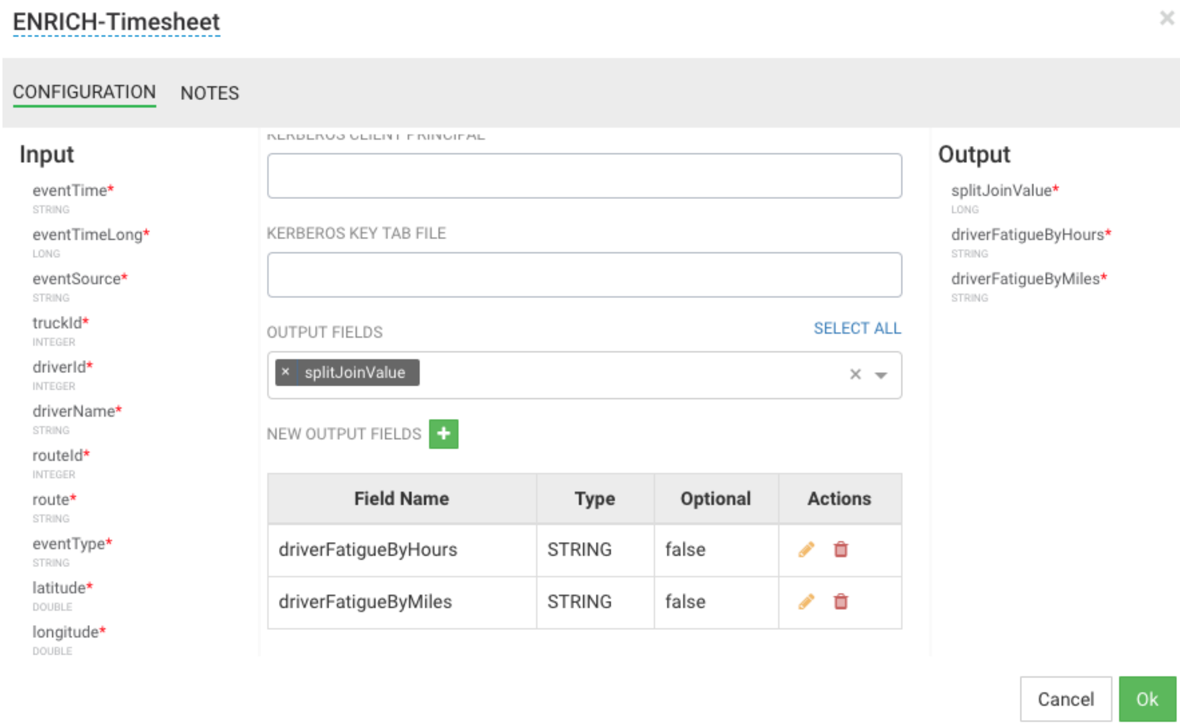
   a. ENRICHEMNT SQL: select hours_logged, miles_logged from timesheet where driverid= ${driverId} and week=${week}

   b. ENRICHMENT OUTPUT FIELDS: driverFatigueByHours, driverFatigueByMiles

   c. SECURE CLUSTER: false

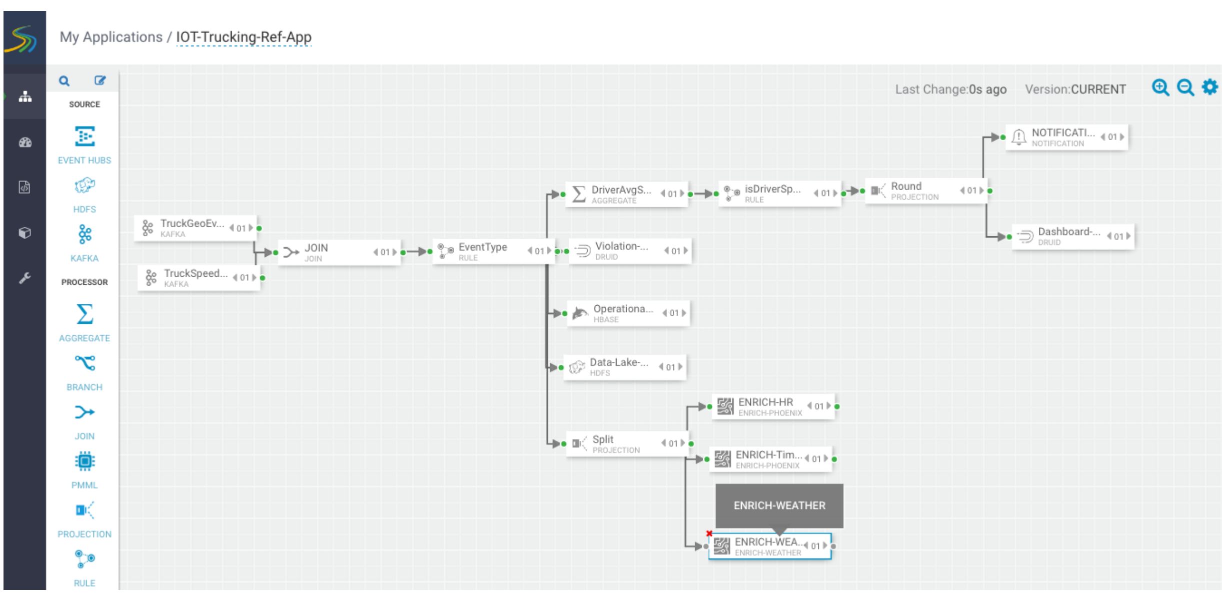
   d. OUTPUT FIELDS: Select the splitJoinValue field.

   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.

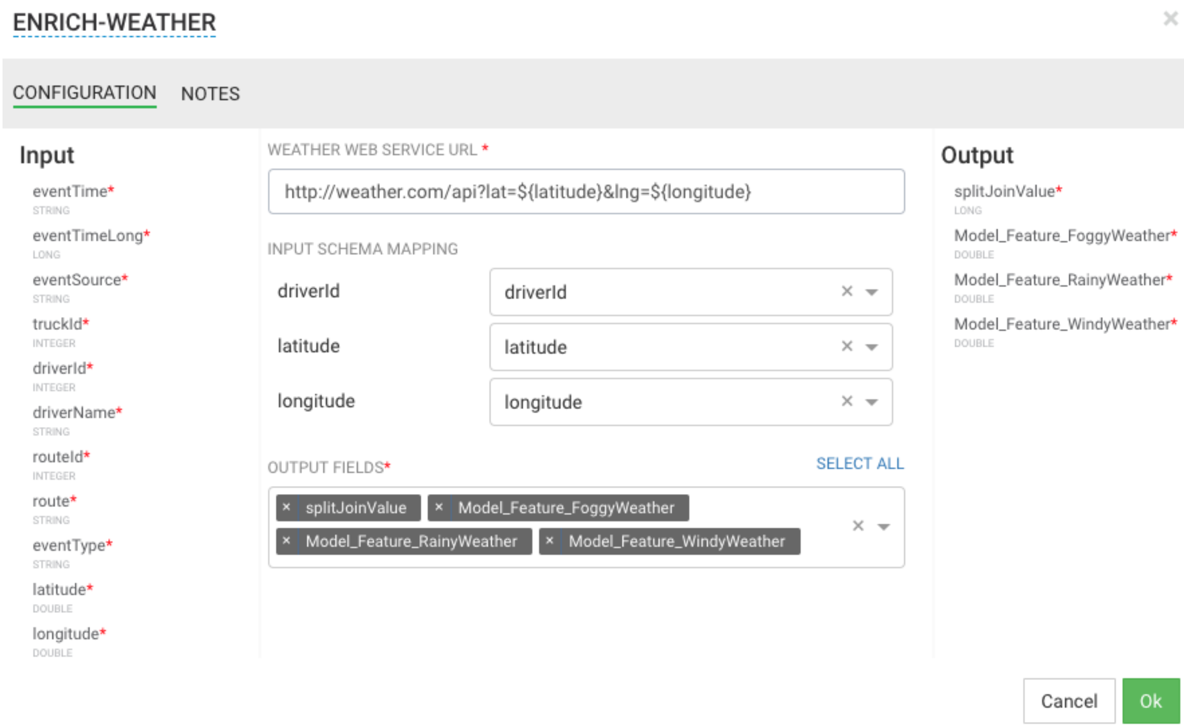


6. Configure the weather process with the following parameters (currently the weather processor is just a stub that generates random normalized weather info).

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

   b. INPUT SCHEMA MAPPINGS: Leave defaults

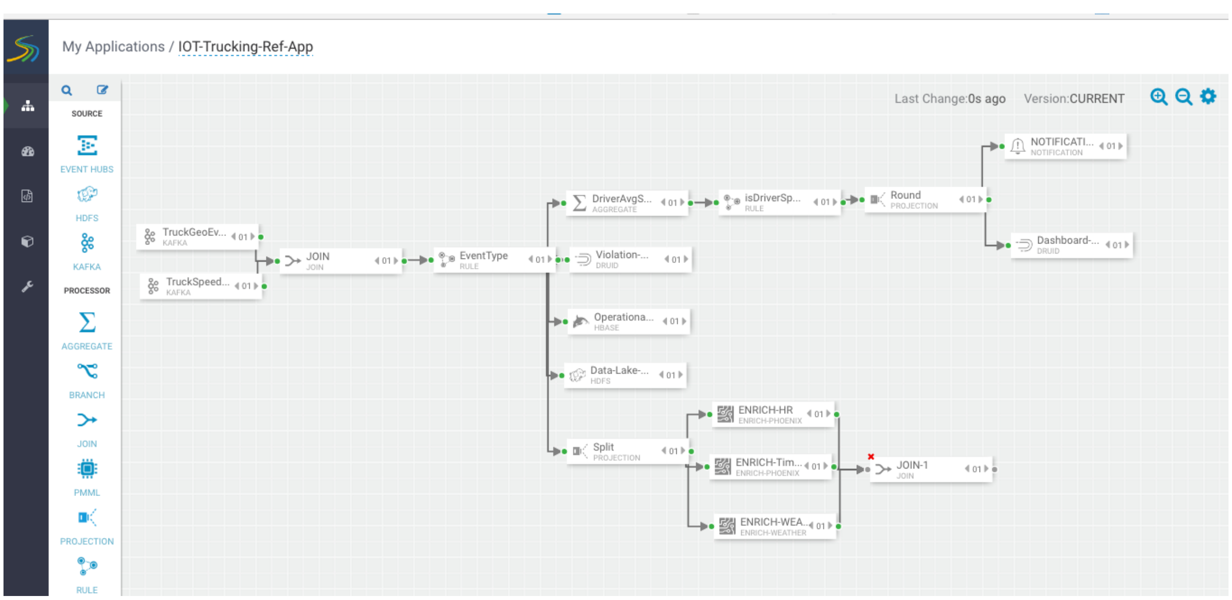
   c. OUTPUT FIELDS: Select the splitJoinValue and the three model enriched features



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

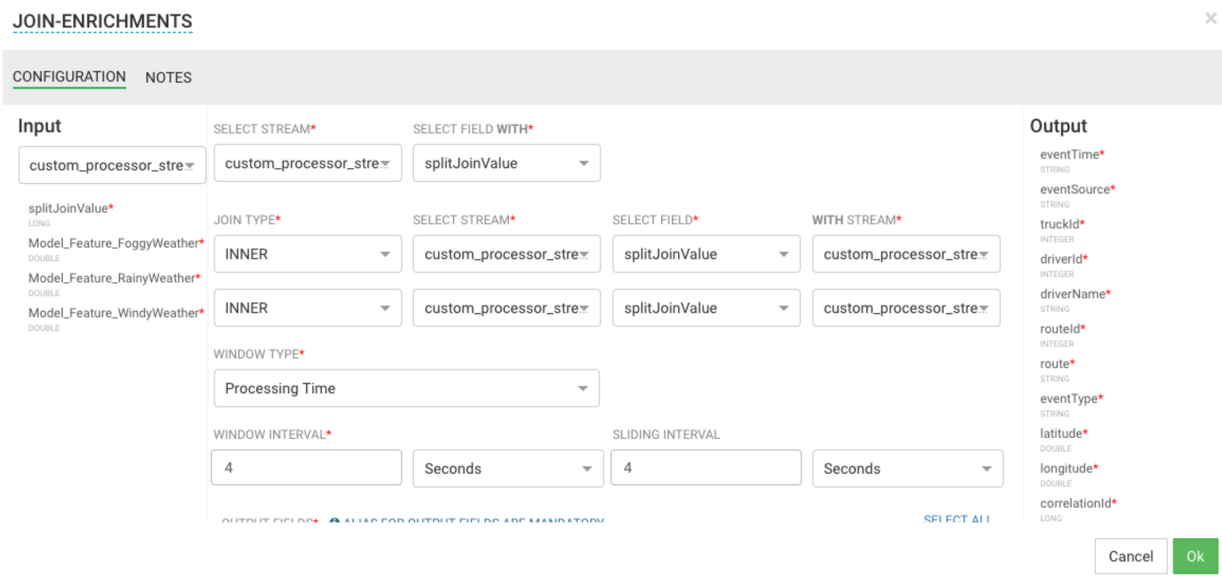
Steps for Rejoining the Three Enriched Streams

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.



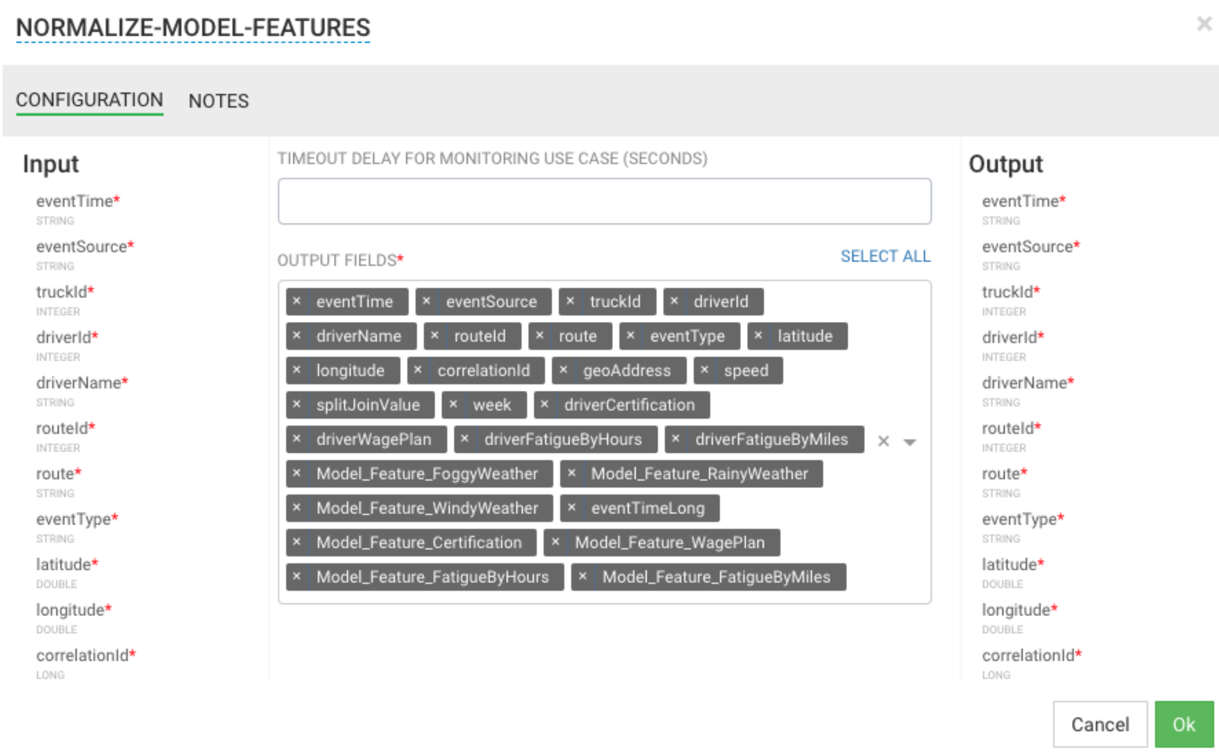
2. Configure the join processor like the following where you use the joinSplitValue to join all three streams.

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



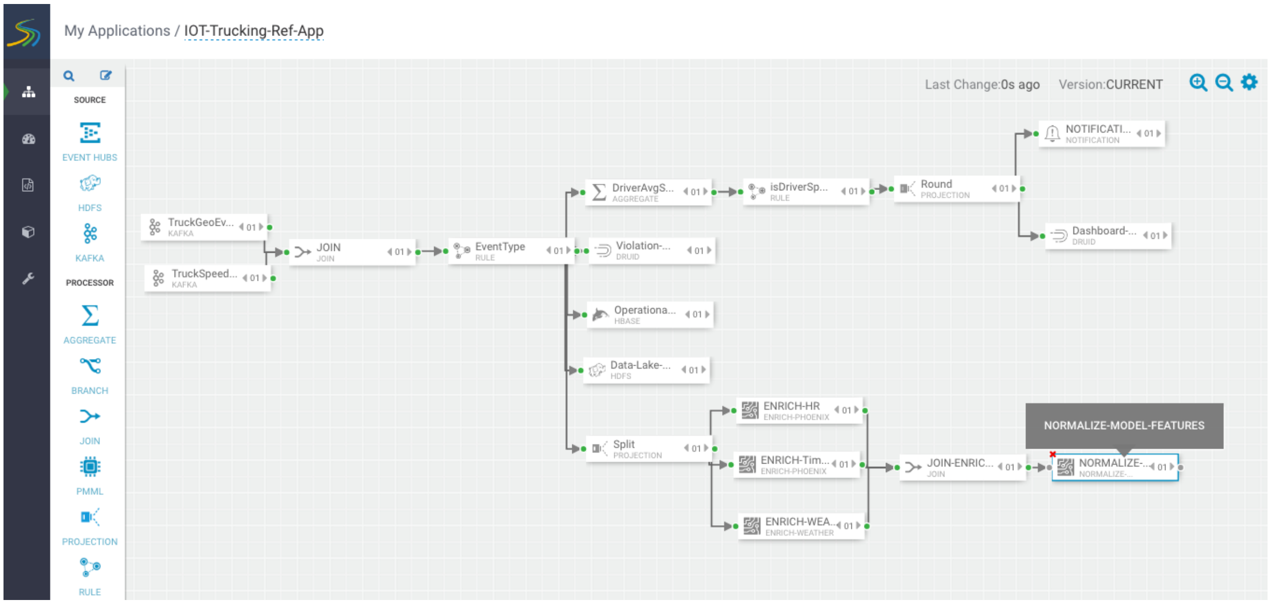
3. Now that you have joined three enriched streams, normalize the data into the format that the model expects by dragging the "NORMALIZE-MODEL-FEATURES" custom processor that you added to the canvas.

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



Result

Your flow looks similar to the following.

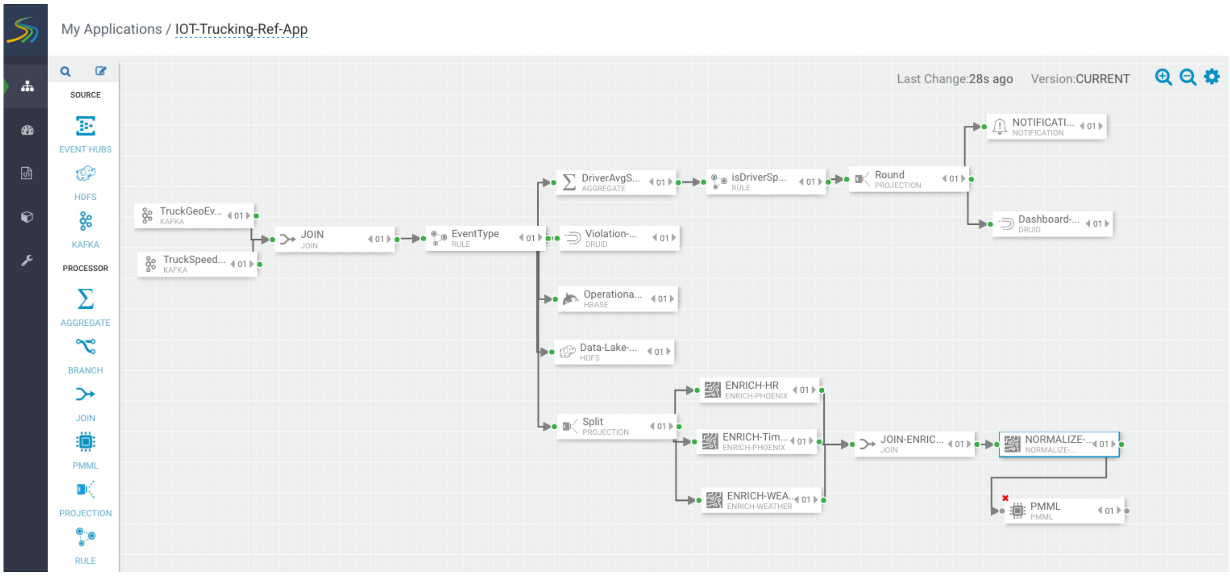


Score the Model Using the PMML Processor and Alert

Now you are ready to score the logistical regression model.

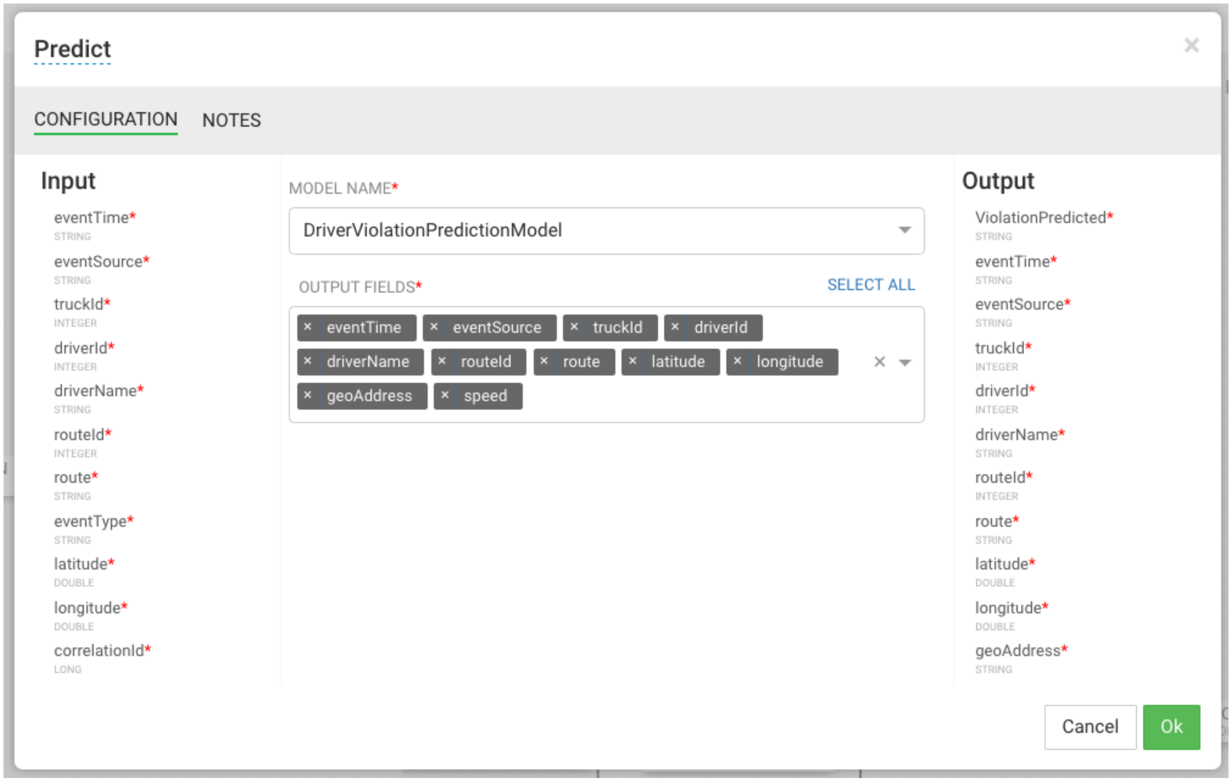
Procedure

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

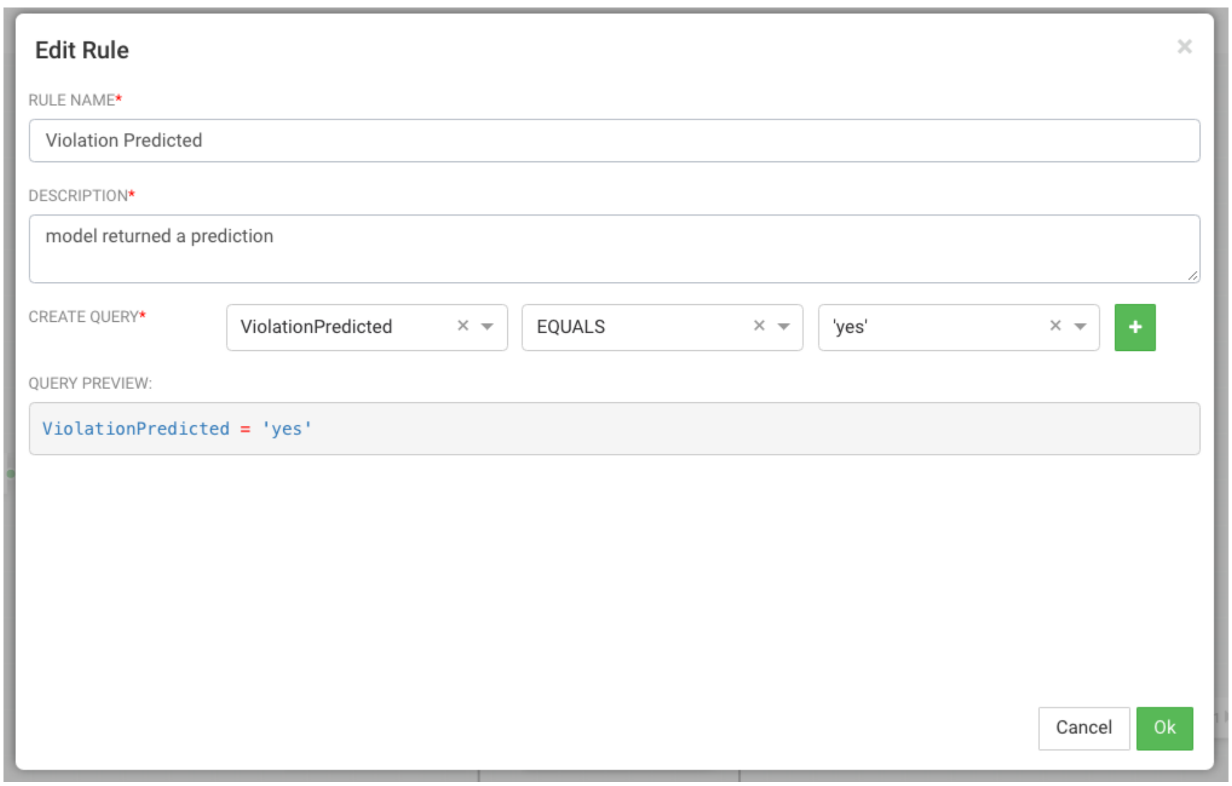


2. Configure the PMML processor like the following by selecting the DriverViolationPredictionModel that you uploaded earlier to the **Model Registry**.

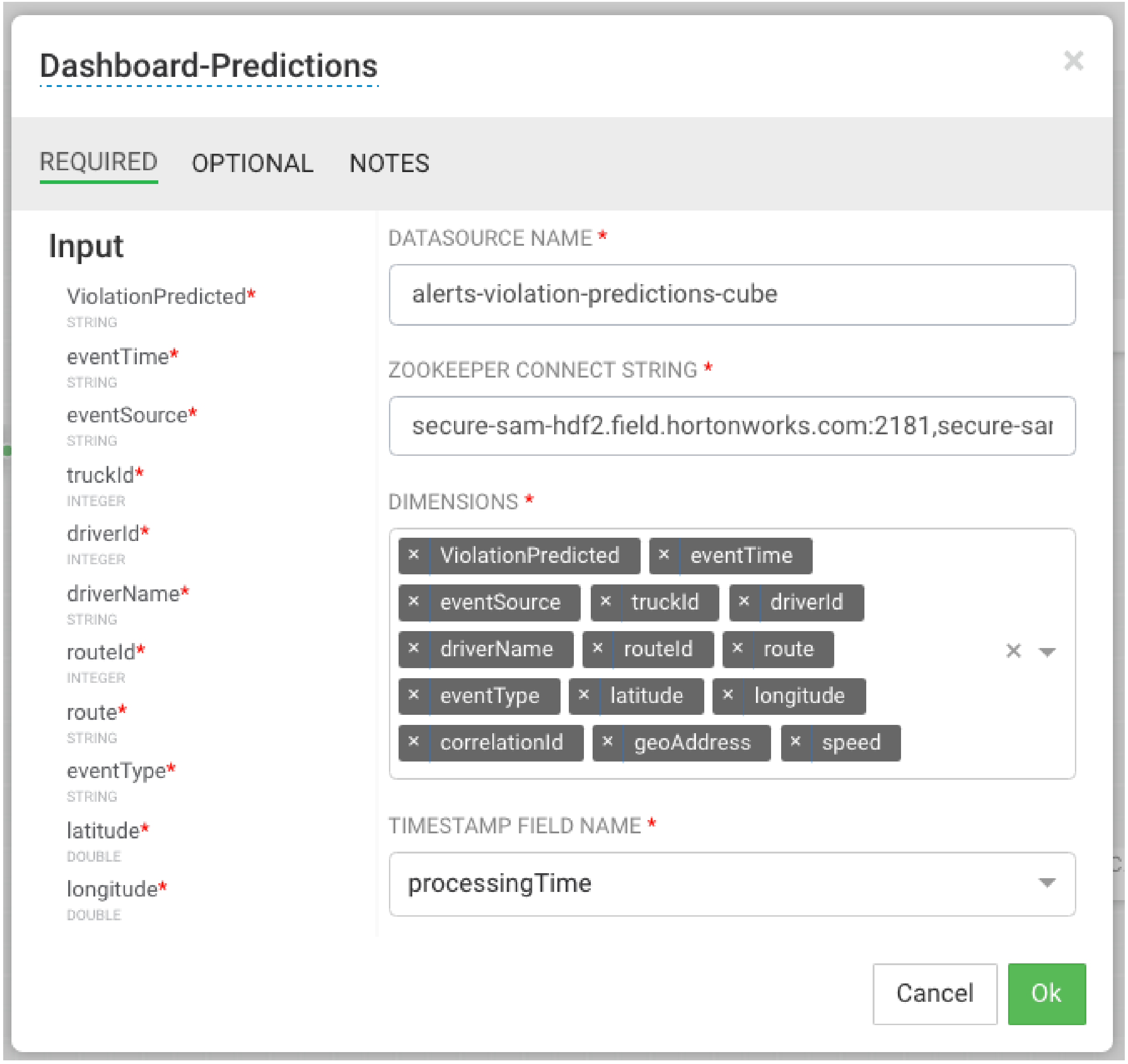
   After this processor executes, a new field called **ViolationPredicted** is added to stream for the result of the prediction. In output fields, select all the contextual fields you want to pass on including the model value result.



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:

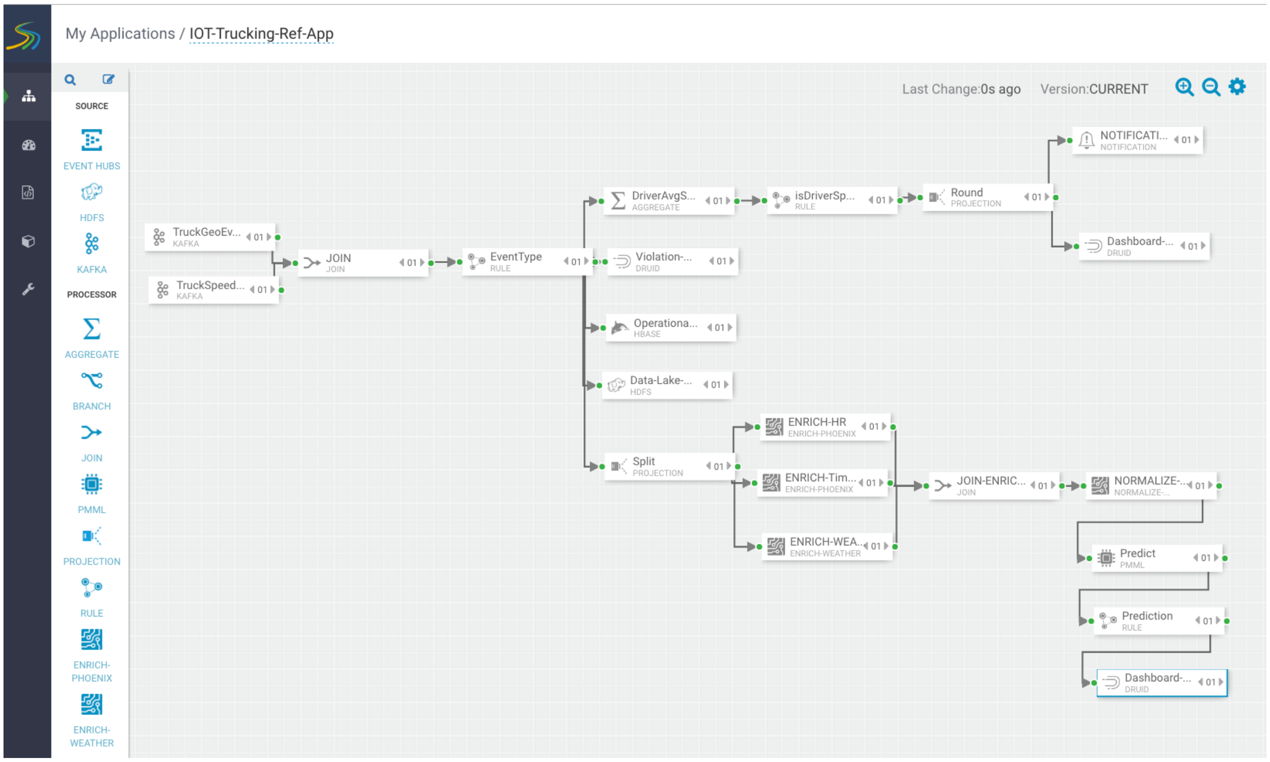


4. If a violation is predicted, send it to a Druid to display on a dashboard. Drag the Druid processor to canvas and configure.
5. Stream the events into a cube called **alerts-violation-predictions-cube**.



Results

The final flow looks like the following:



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:

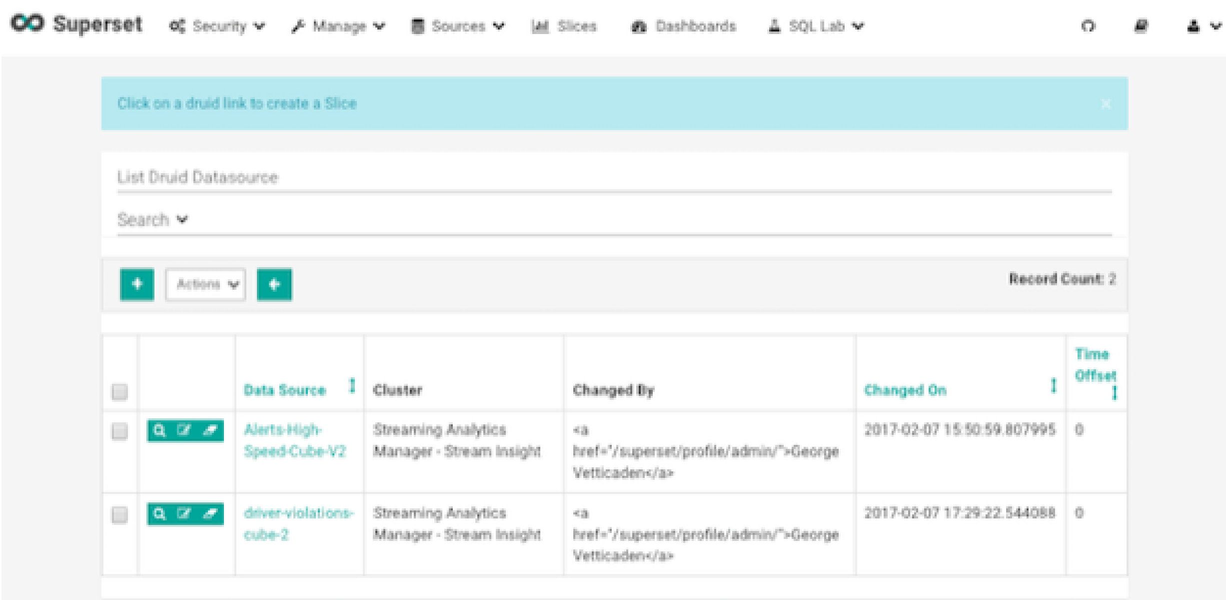
When a SAM app streams data to a new cube using the Druid processor, it will take about 30 minutes for the cube to appear in Superset. This is because Superset has to wait for the first segment to be created in Druid. After the cube appears, users can analyze the streaming data immediately as it is streaming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

Creating Insight Slices

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

Procedure

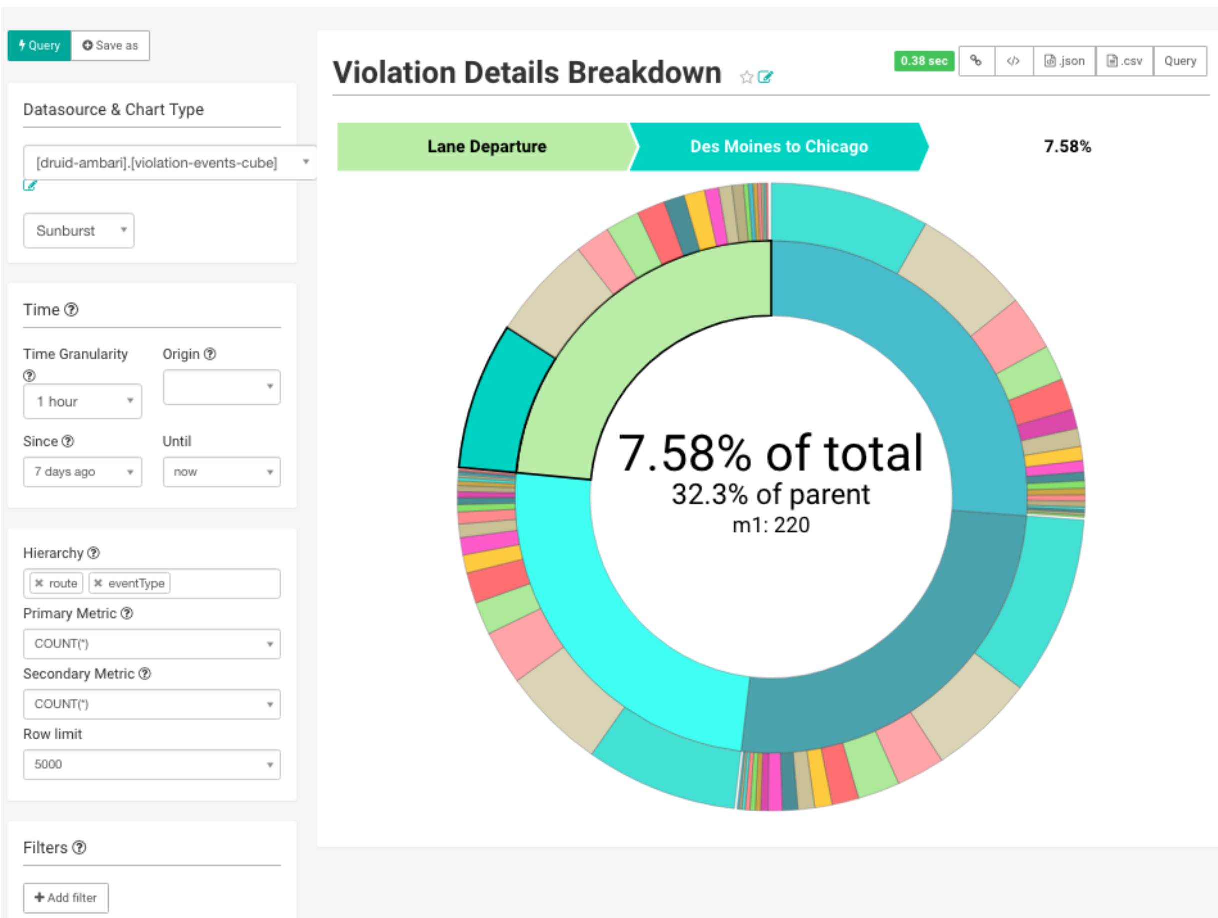
1. Choose Slices on the Menu.
2. Click + to create a new Slice.
3. Select the Druid Data Source that you want to use for the new visualization:



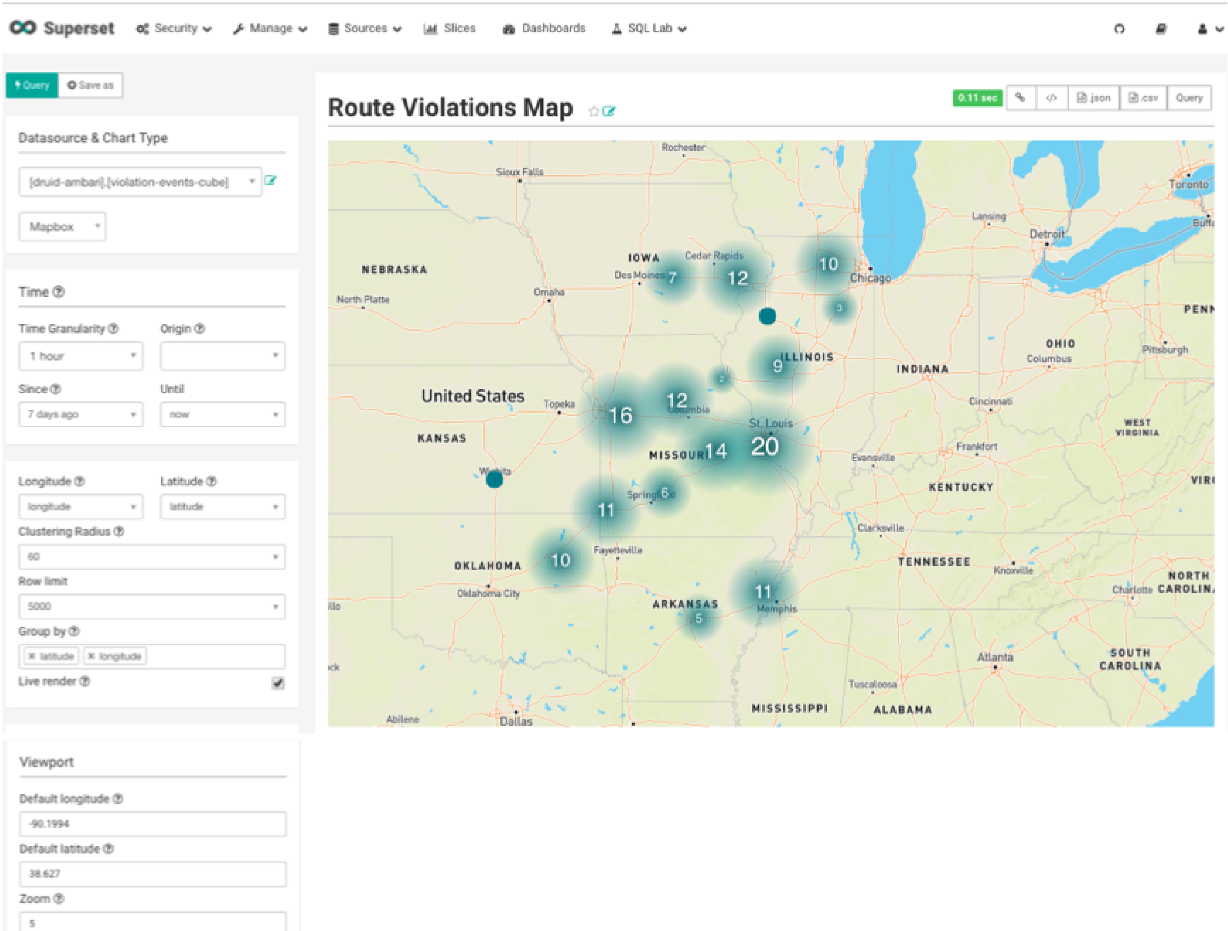
4. Select a Chart Type from the menu.

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

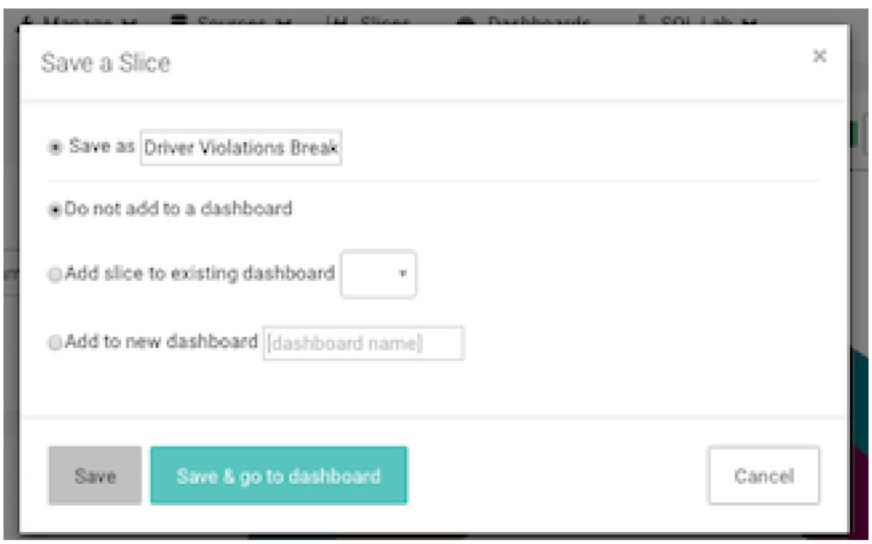
Configure the chart and click Execute Query



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.

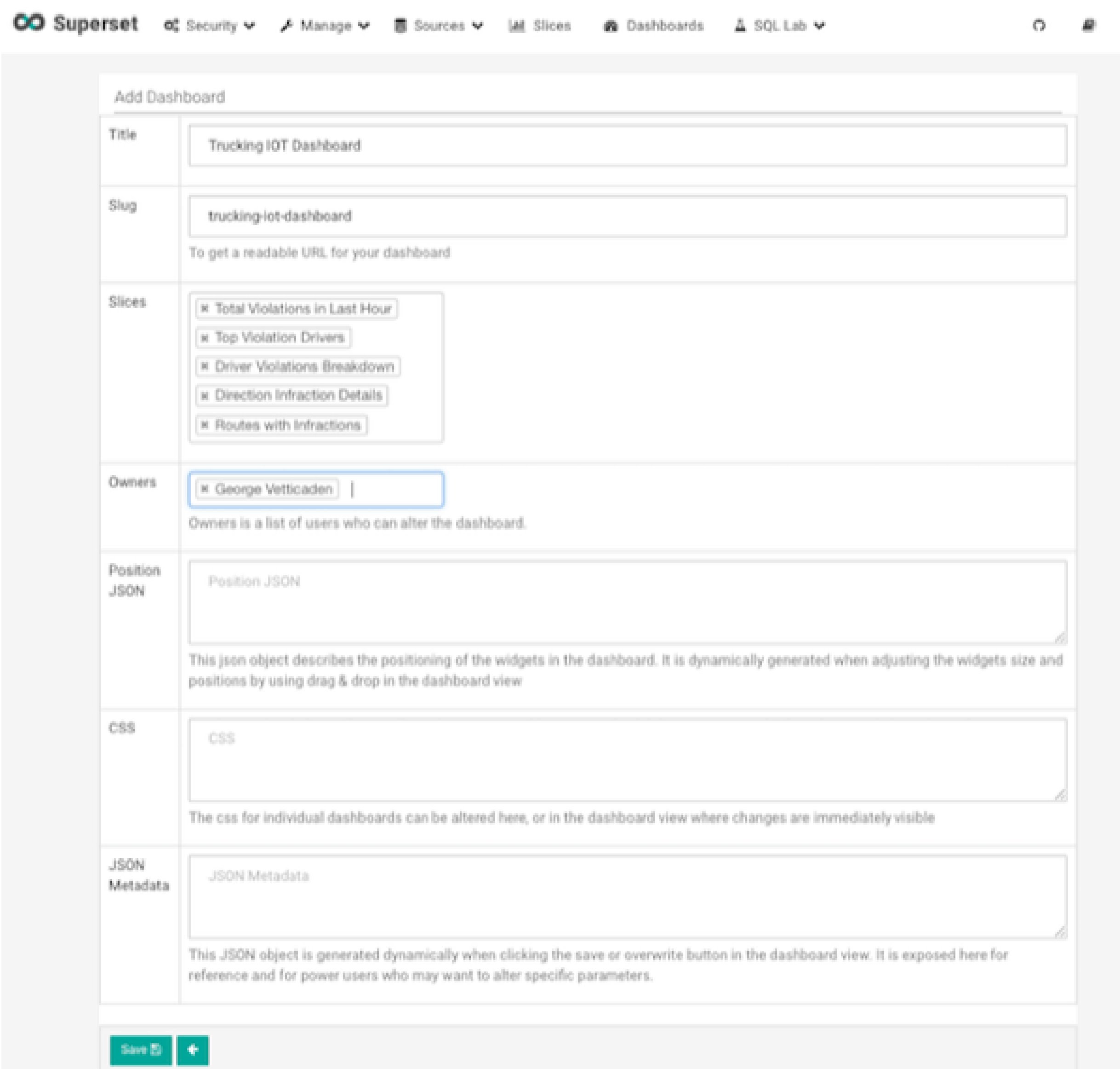


Adding Insight Slices to a Dashboard

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

Procedure

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.

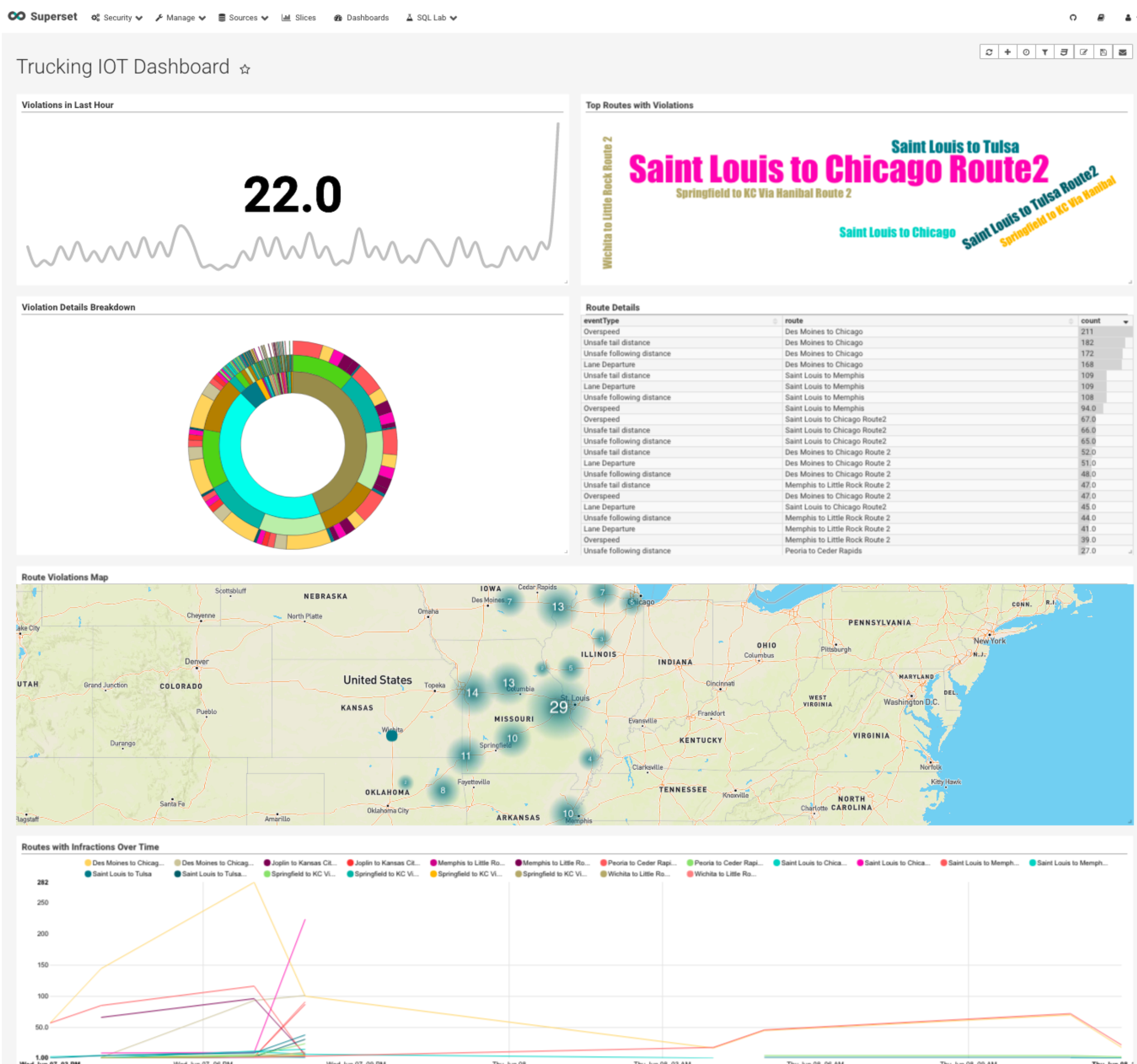
Dashboards for the Trucking IOT App

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

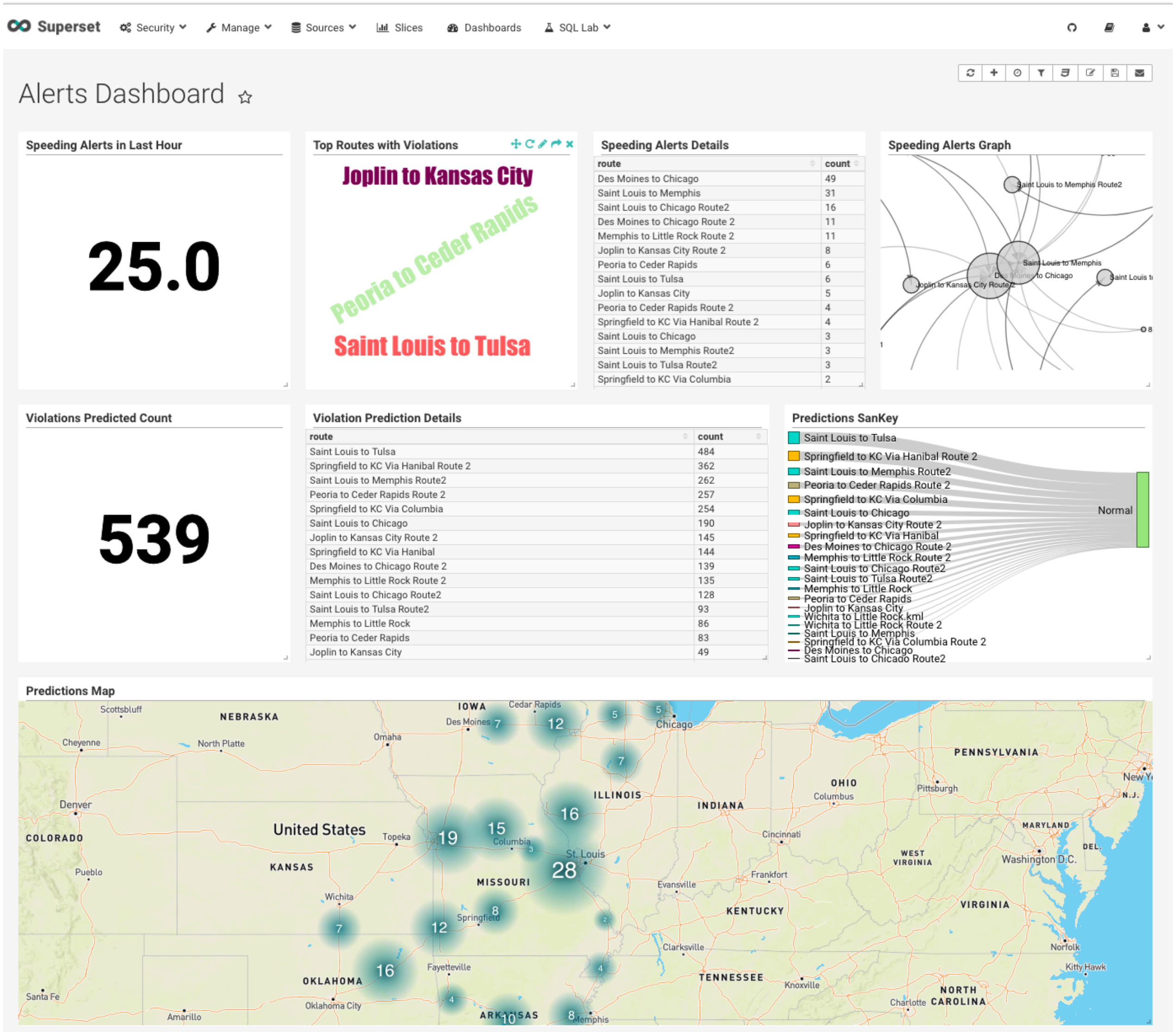
- violation-events-cube
- alerts-speeding-drivers-cube
- alerts-violation-predictions-cube

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

IoT Dashboard



Alerts Dashboard



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.

Four Test Cases using SAM's Test Mode

Test Case 1: Testing Normal Event with No Violation Prediction

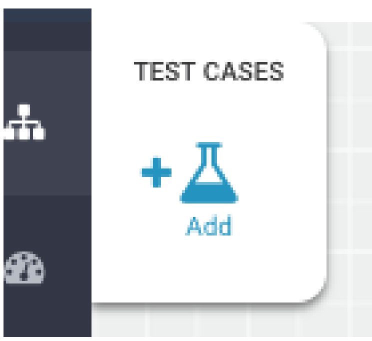
The Assertions of this test case are the following:

- 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.

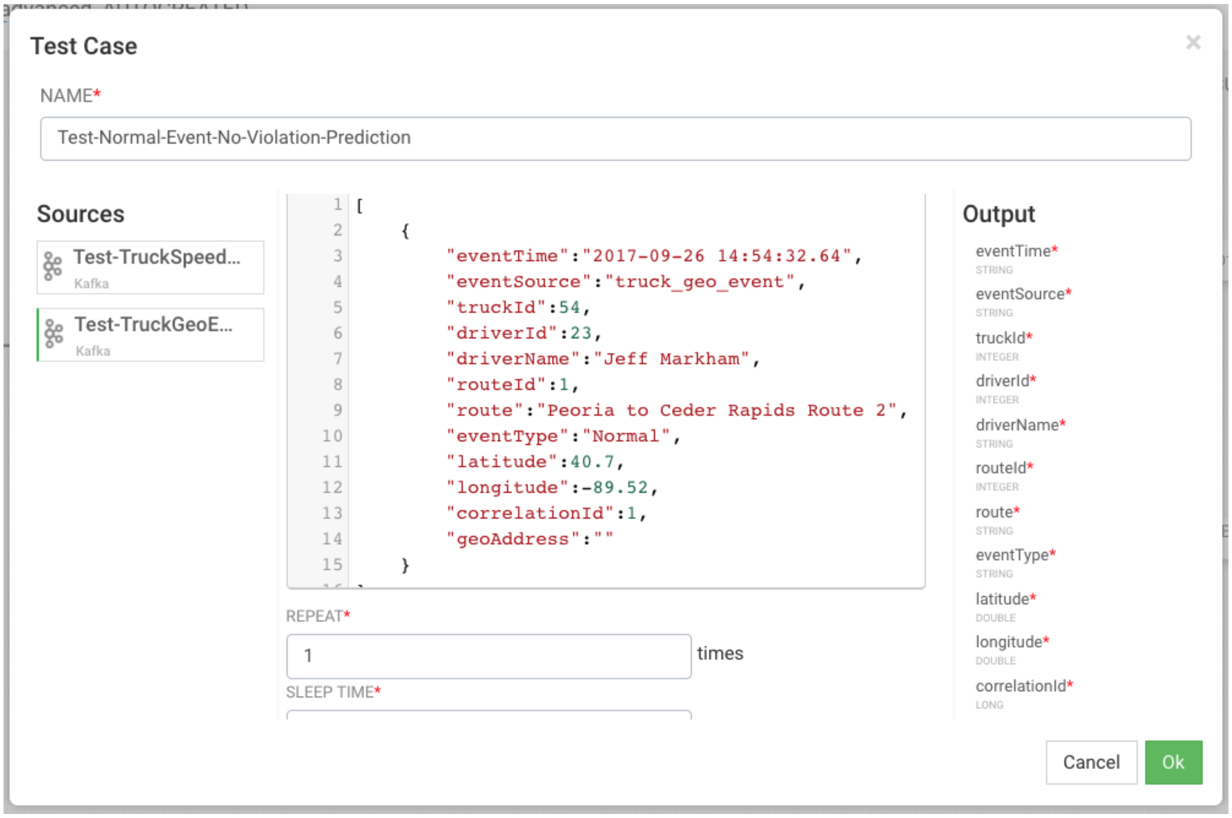
Follow the below steps to create this test case for the reference app in Edit Mode. (If you ran the test utility, these 4 test cases will already be created for you.)

1. Click “TEST” Mode

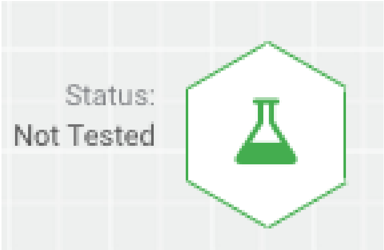
2. Click Add Test Case



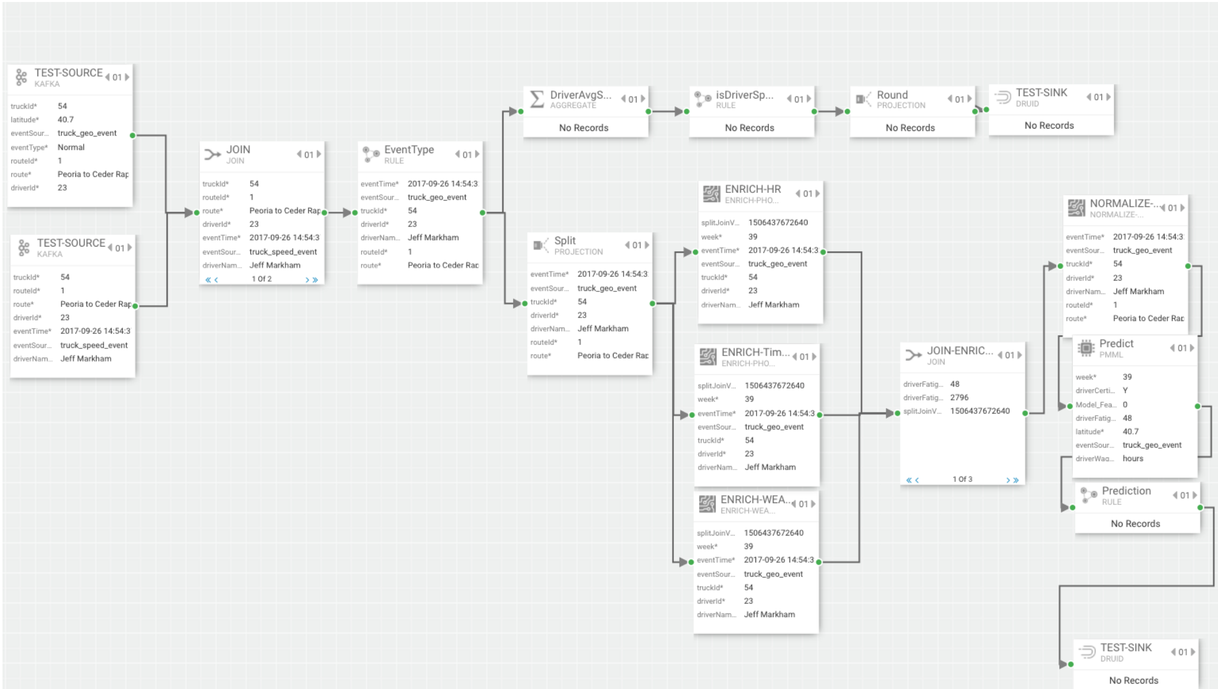
3. Provide Test Case details. Provide a name for test case, test data for TruckGeoEvent and test data for TruckSpeedEvent.



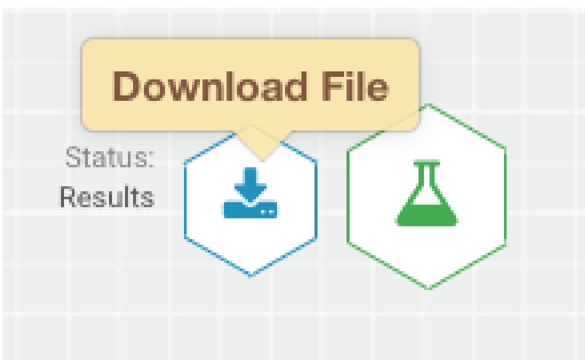
4. Execute the Test Case.



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



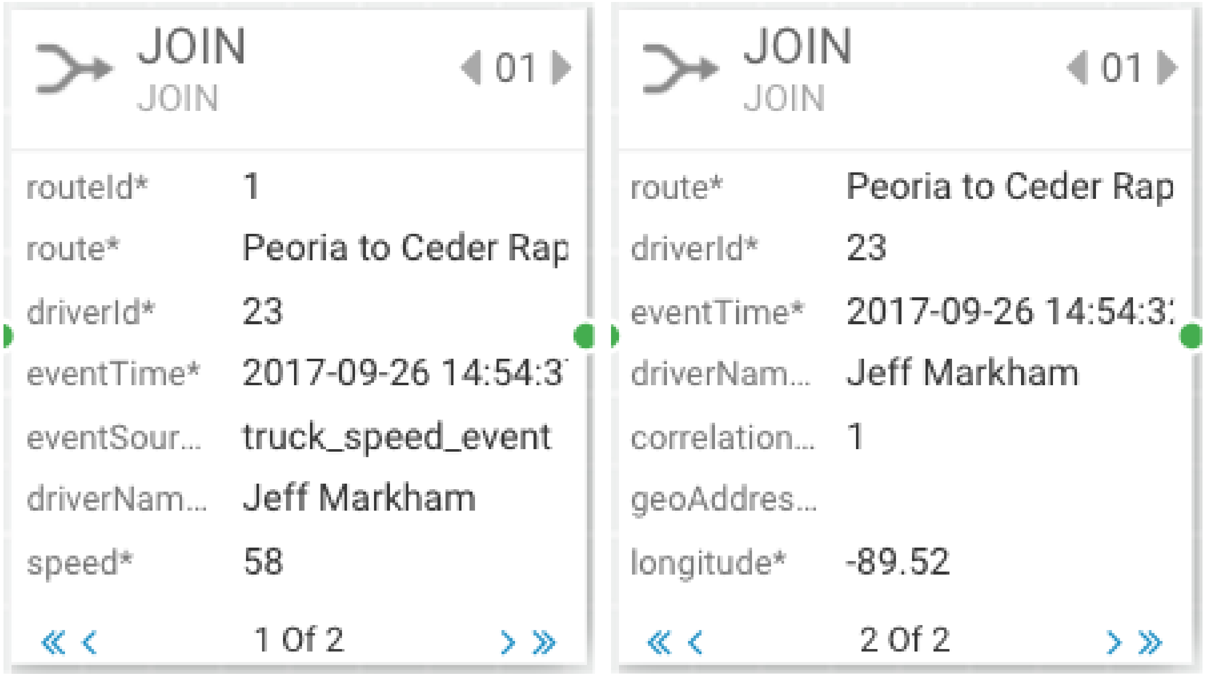
6. Download the test case results.



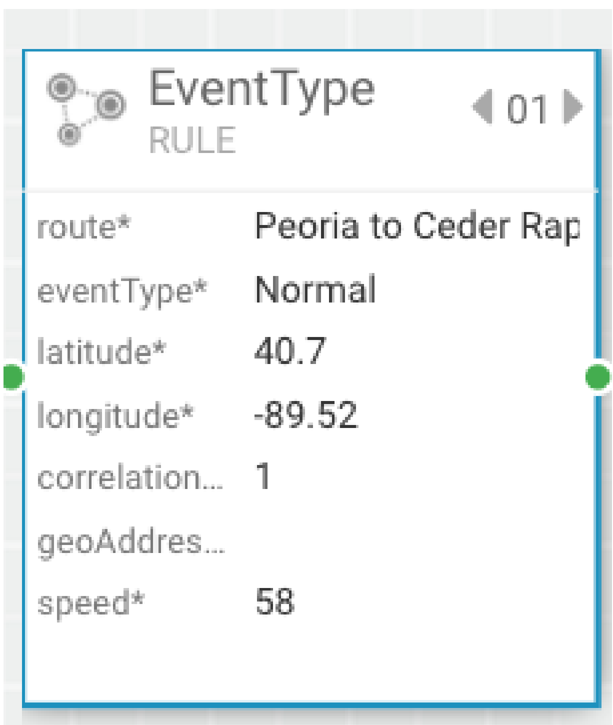
Analyzing Test Case 1 Results

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.

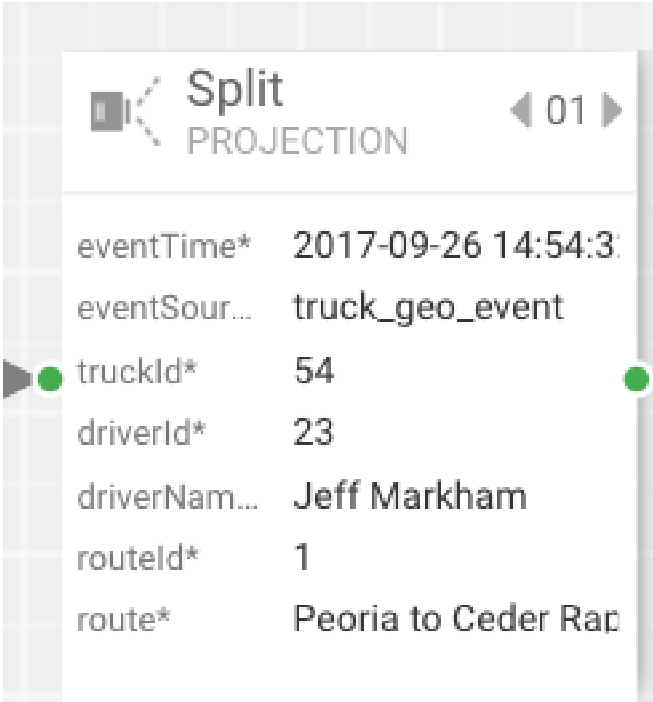
- Assertion 1 is to Validate test data for geo steam and speed stream that are non violations. For this assertion, you would look at the downstream component after the sources. So in this case, it would be the Join component. Use the paging features to see the inputs to the join processor.



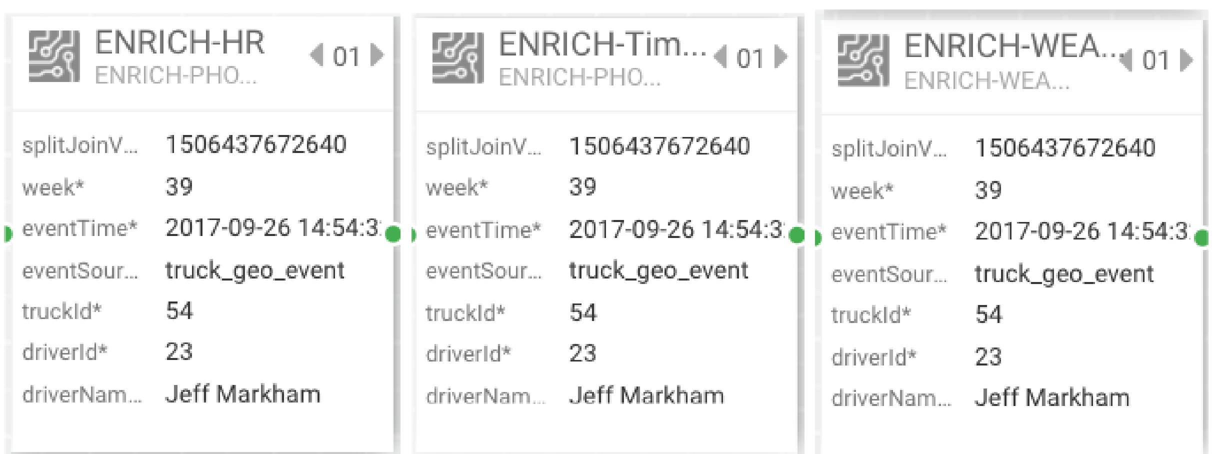
- Assertion 2 is to validate the Join of data between geo stream and speed stream. For this assertion, you would look at the downstream component after the Join. So in this case, it would be the EventType component. Note that you see speed and geo information.



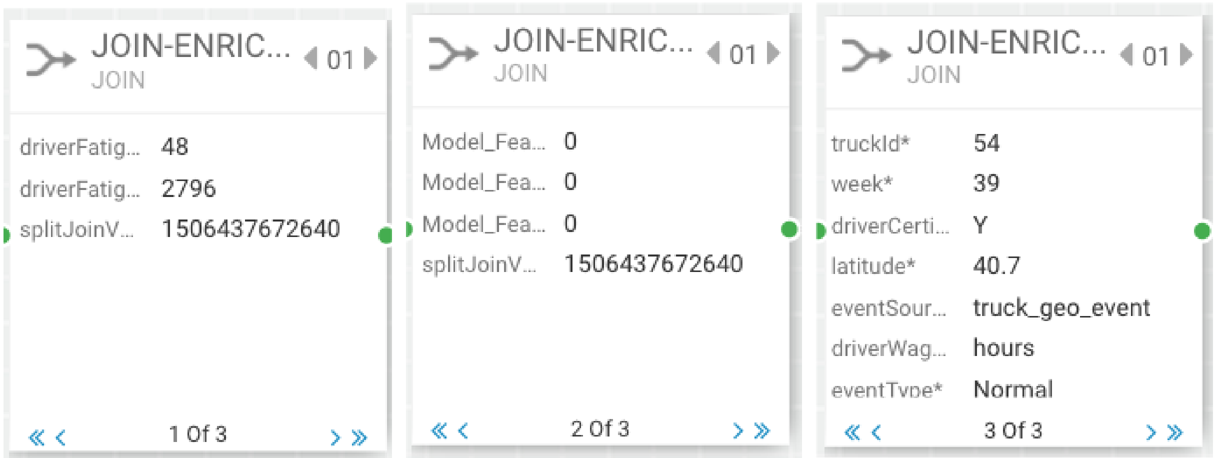
- Assertion 3 is to validate that the filter “EventType” detects that this is a “Non Violation Event”. View the Split Component.



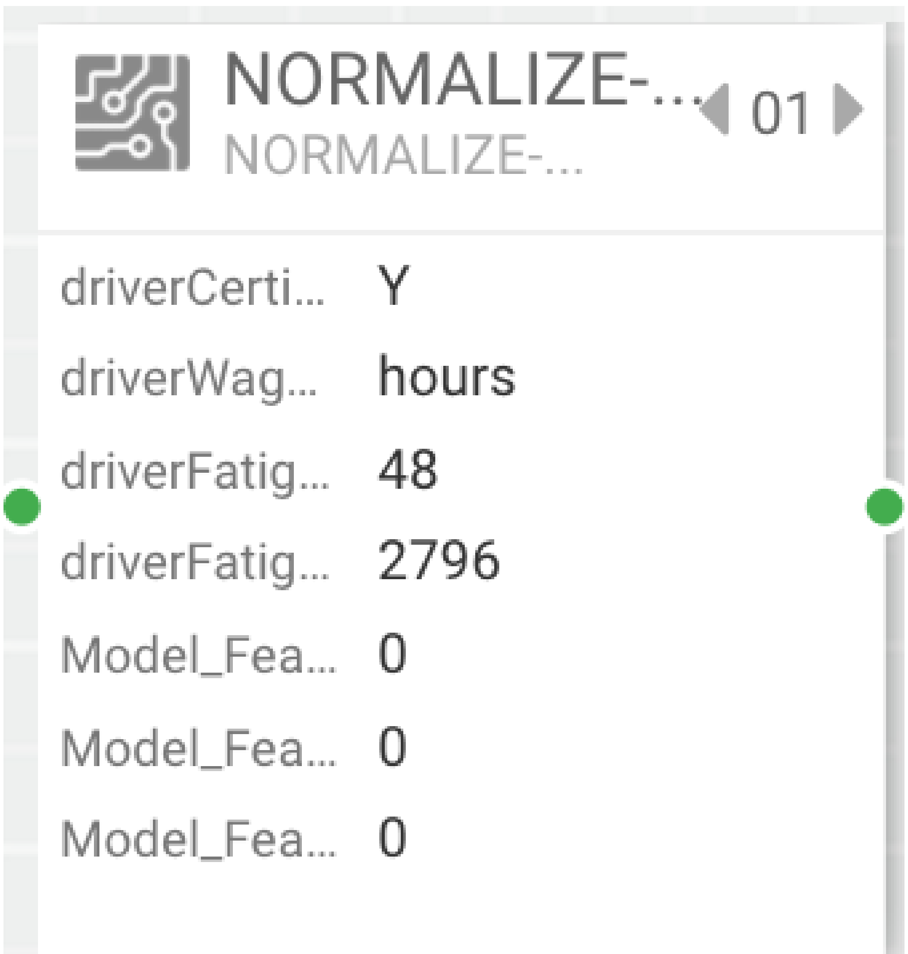
- Assertion 4 is to Validate test data for geo steam and speed stream that are non violations. View the JOIN-ENRICHMENT component



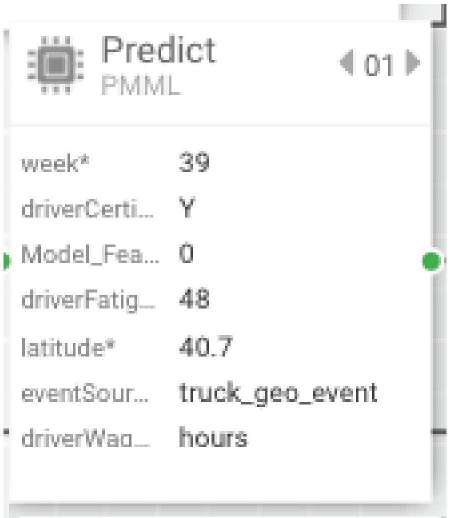
- Assertion 5 is to validate that the three enrichments are applied: weather enrichments, timesheet enrichment and HR enrichment. Use the paging features to page through the three enrichment outputs.



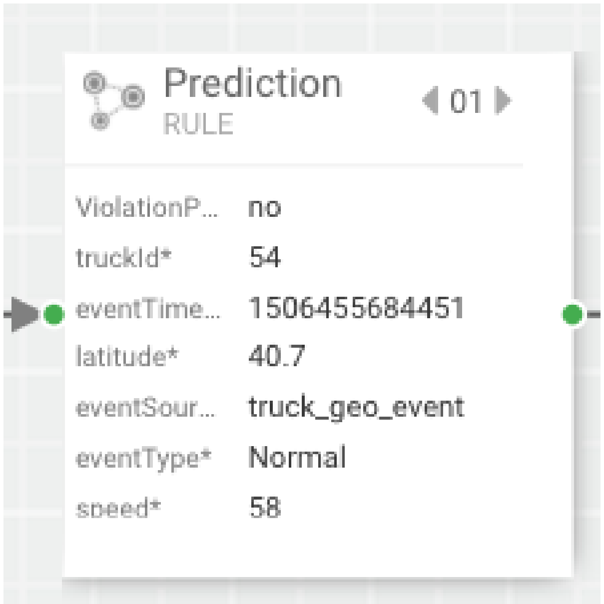
- Assertion 6 is to validate the three enrichment streams are joined into a single stream. View the NORMALIZE component.



- 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.



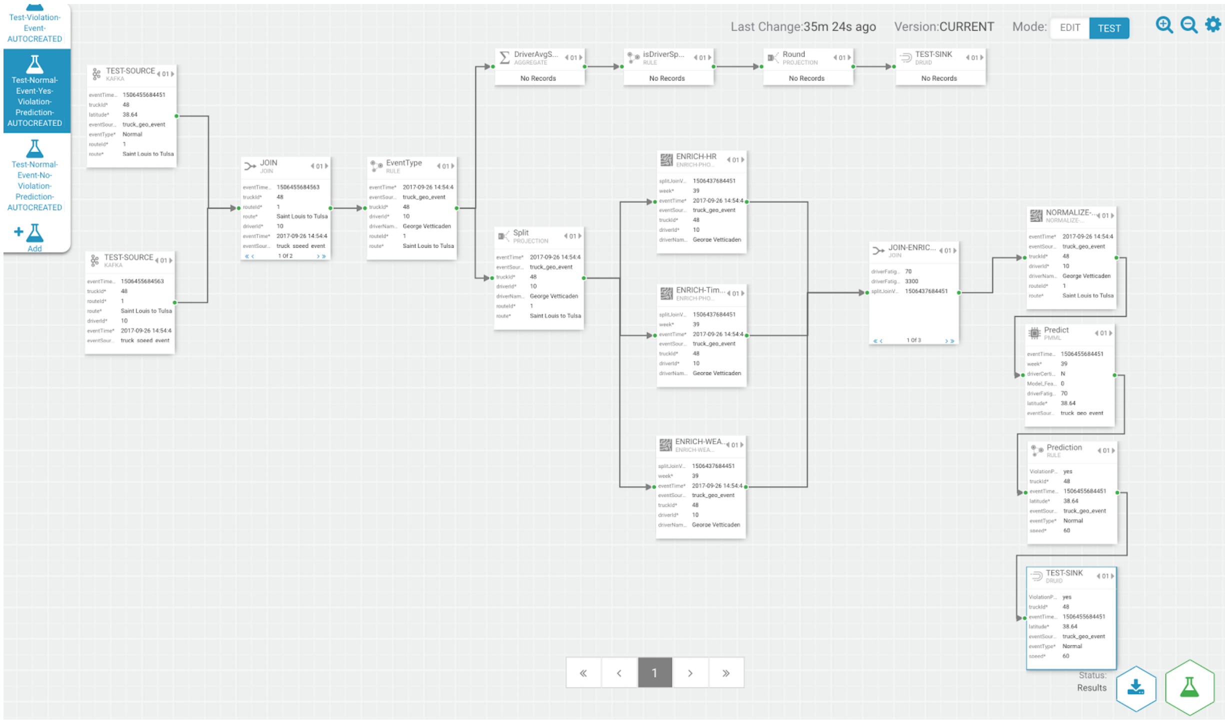
Test Case 2: Testing Normal Event with Yes Violation Prediction

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.

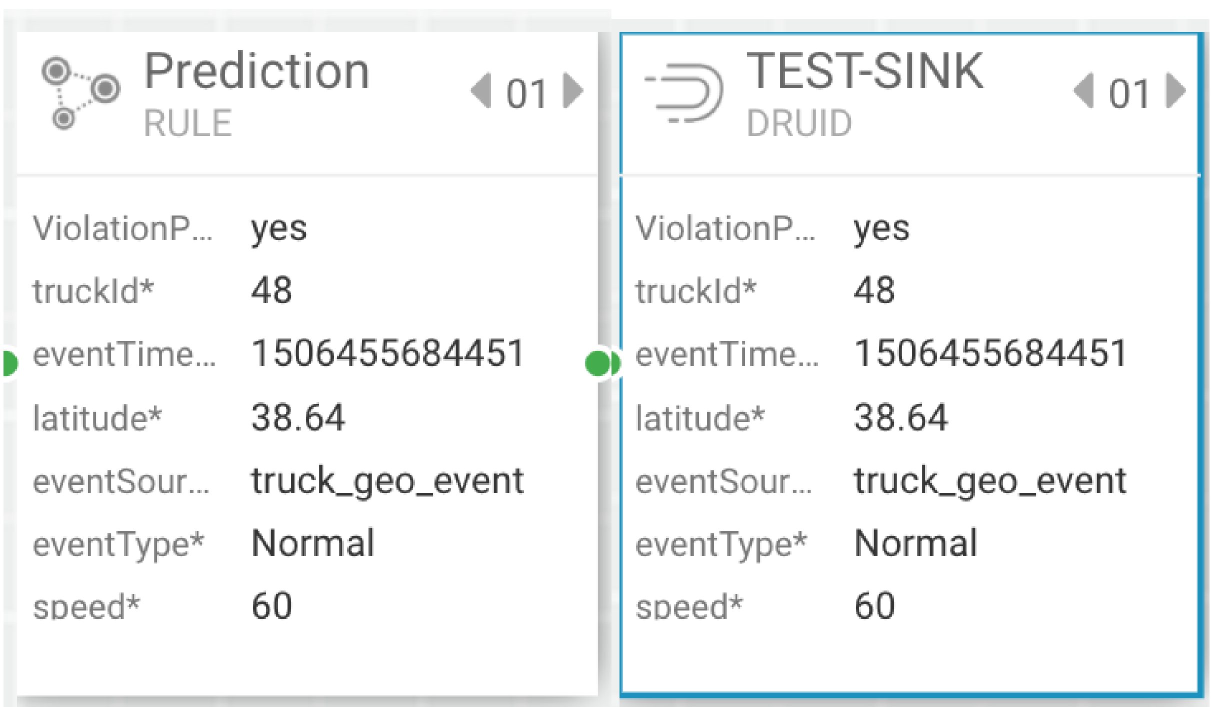
Analyzing Test Case 2 Results

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.



Test Case 3: Testing Violation Event

The Assertions of this test case are the following:

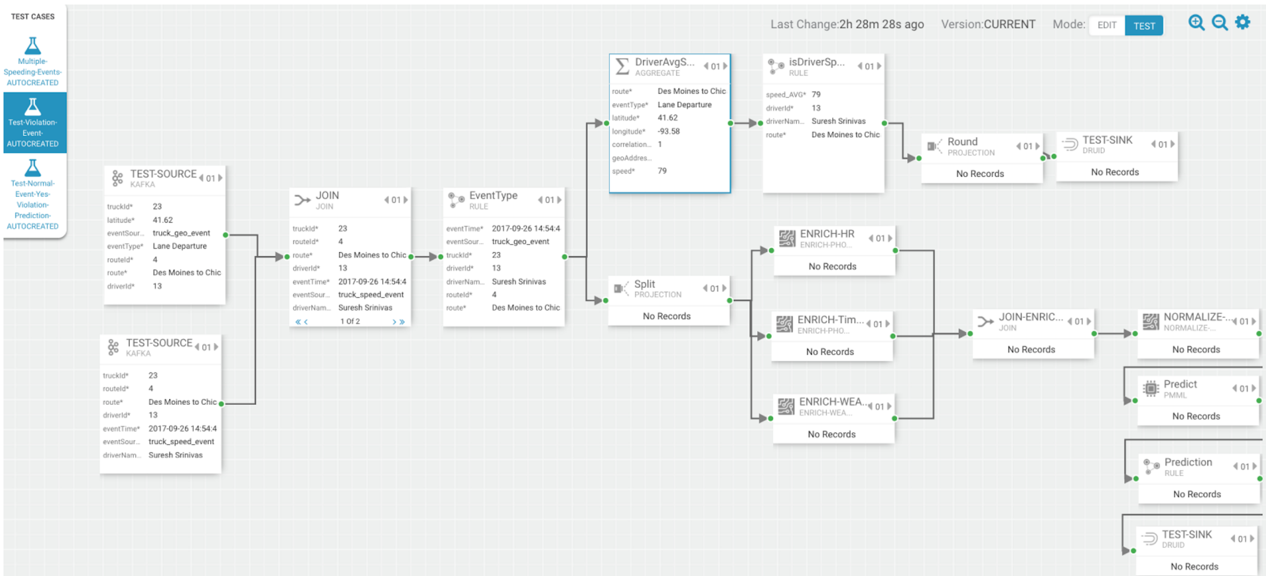
- Assertion 1: Validate test data for geo steam and speed stream that are “violation” events
- Assertion 2: Validate the Join of data between geo stream and speed stream
- Assertion 3: Validate that the filter “EventType” detects that this is a “Violation Event”

- 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.

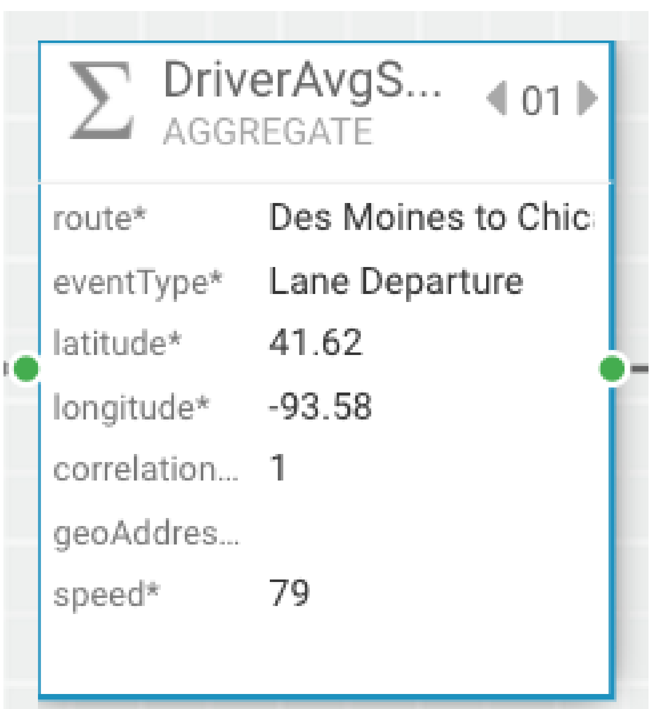
Create a test named "Test-Violation-Event", use the following test data for TruckGeoEvent and use the following test data for TruckSpeedEvent.

Analyzing Test Case 3 Results

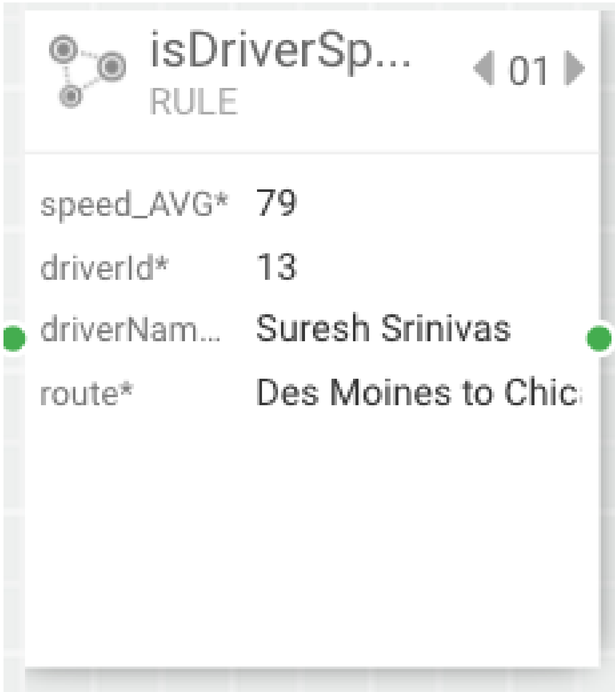
The output of the test case should look something like the following:



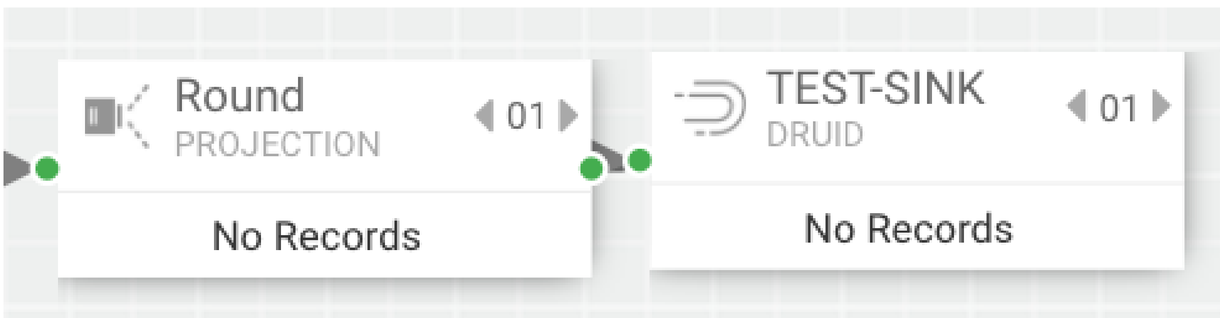
- Assertion 3 is to validate that the filter "EventType" detects that this is a "Violation Event" and Assertion 4 is to validate that the inputs to the aggregate speed processor should be 1 event within the window. View the DriverAvSpeed component to validate these assertions:



- 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.



Test Case 4: Testing Multiple-Speeding-Events

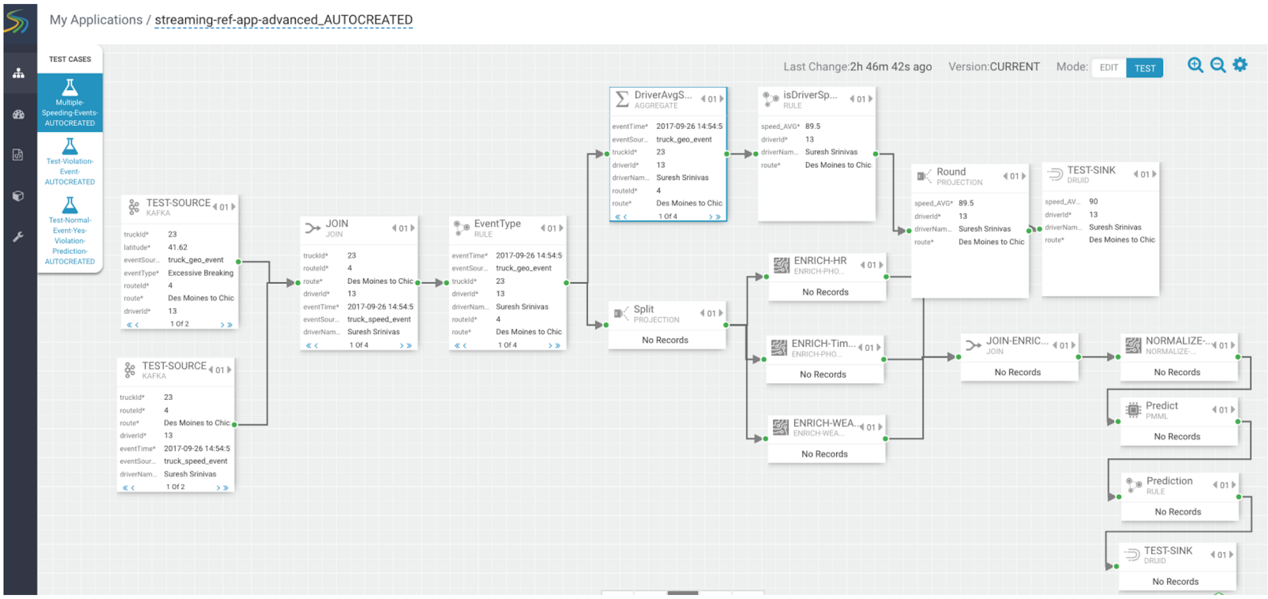
The assertions of this test case are the following:

- 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.

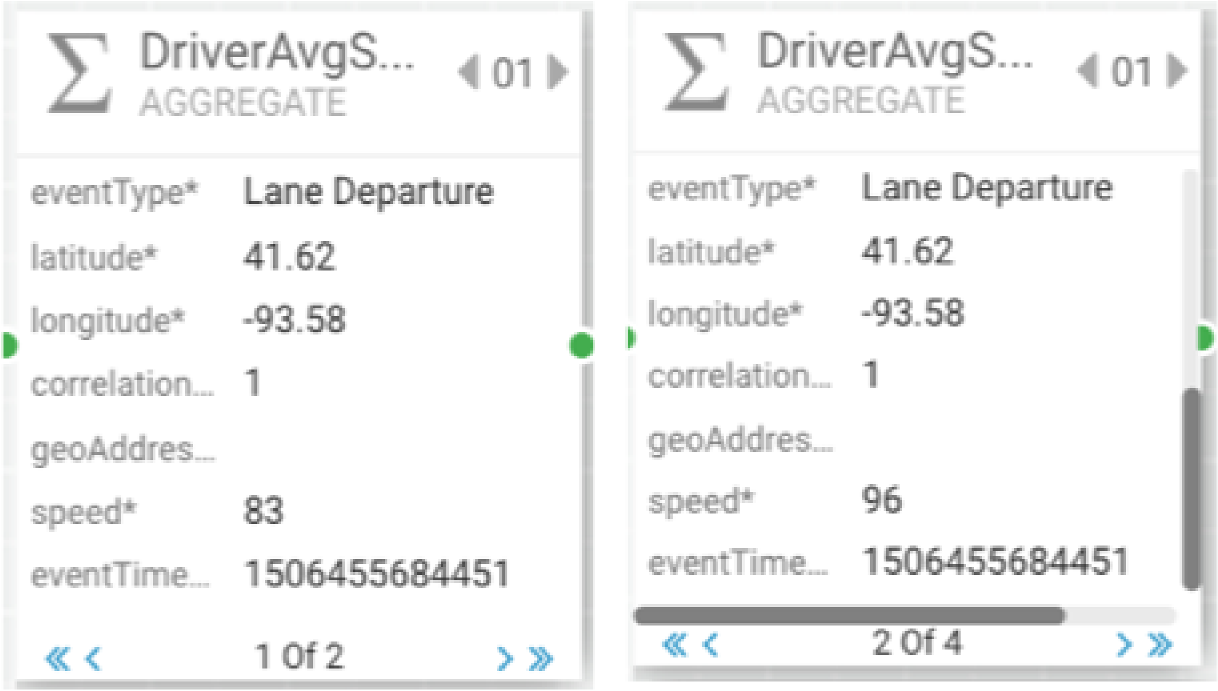
Create a test named "Test-Multiple-Speeding-Events", use the following test data for TruckGeoEvent and use the following test data for TruckSpeedEvent.

Analyzing Test Case 4 Results

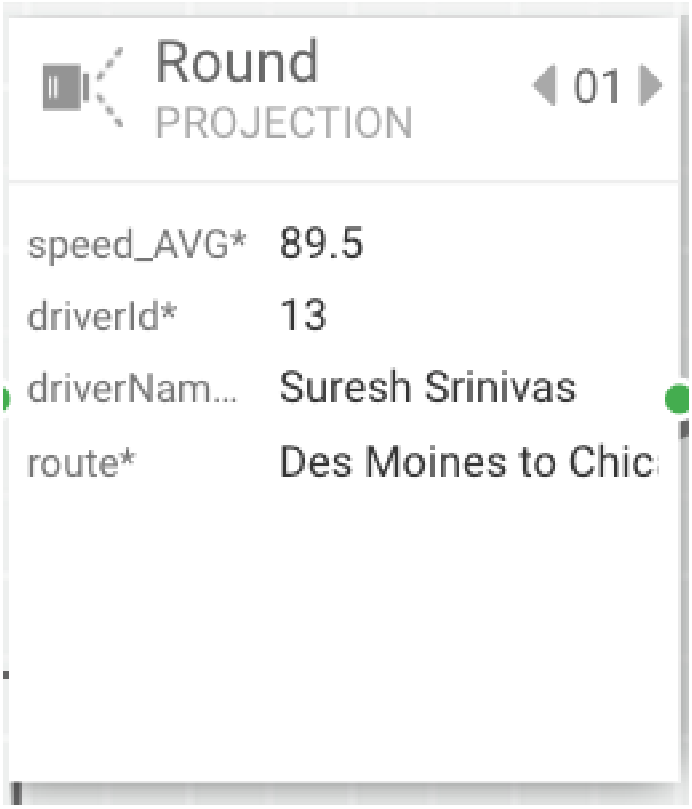
The output of the test case should look something like the following:



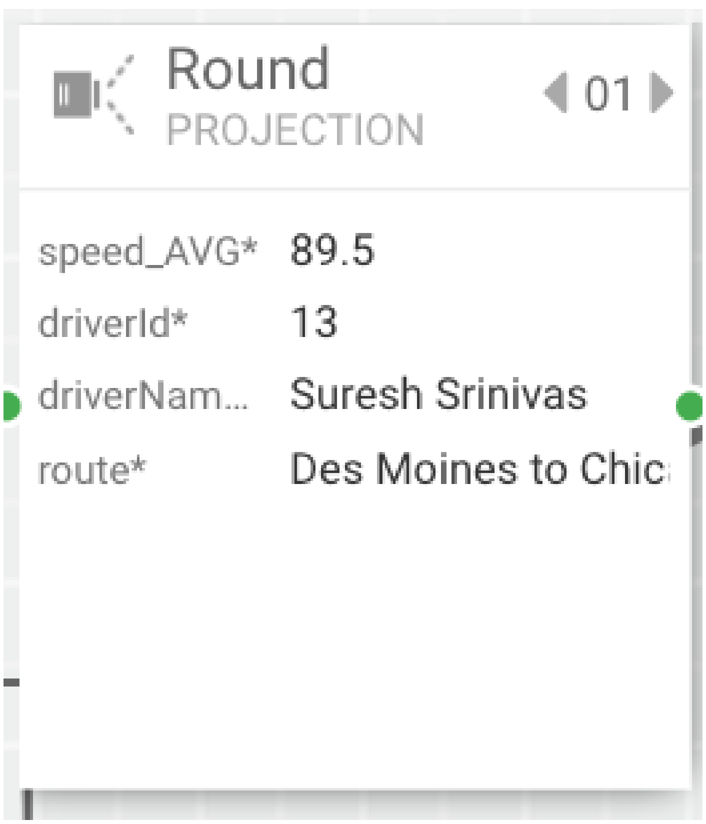
- Assertion 4 is to validate the inputs of the window should be two events (geo/speed 1 with speed of 83, geo/speed 2 with speed of 96). View the two events in the Join processor (use the paging feature to see the events)



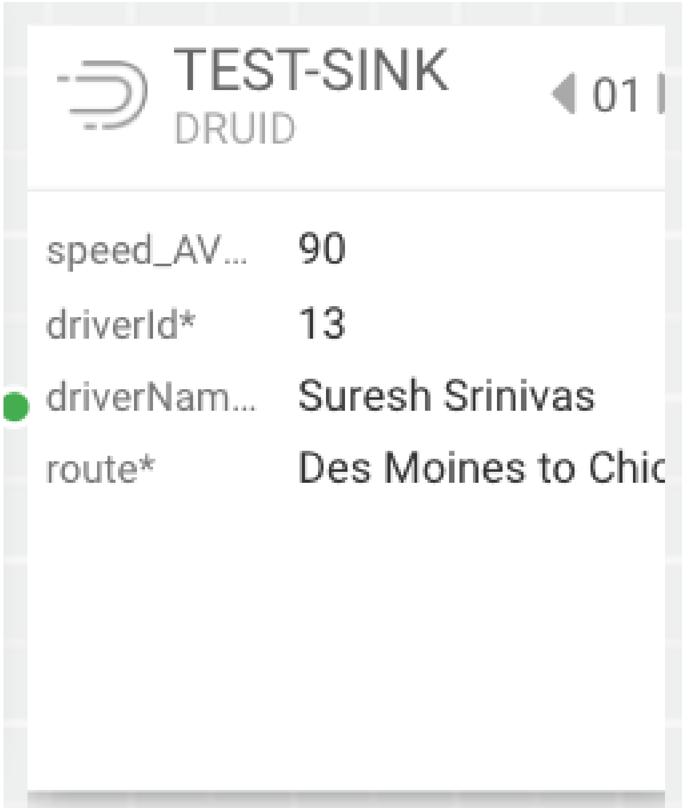
- Assertion 5 is to validate the result of the DriverAvgSpeed aggregate processor should be one event that represents the average of 83 and 96…89.5. View the Round Projection processor.



- 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.



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:

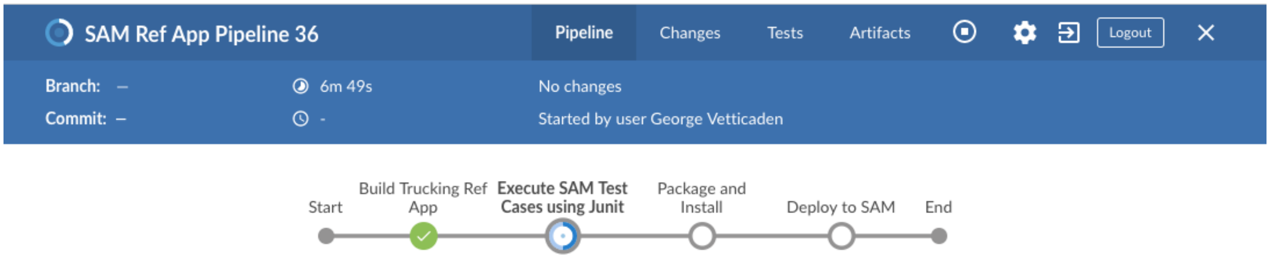
- testNormalEventNoViolationPrediction
- testNormalEventYesViolationPrediction
- testViolationTruckingEvents
- testMultipleSpeedingEvents

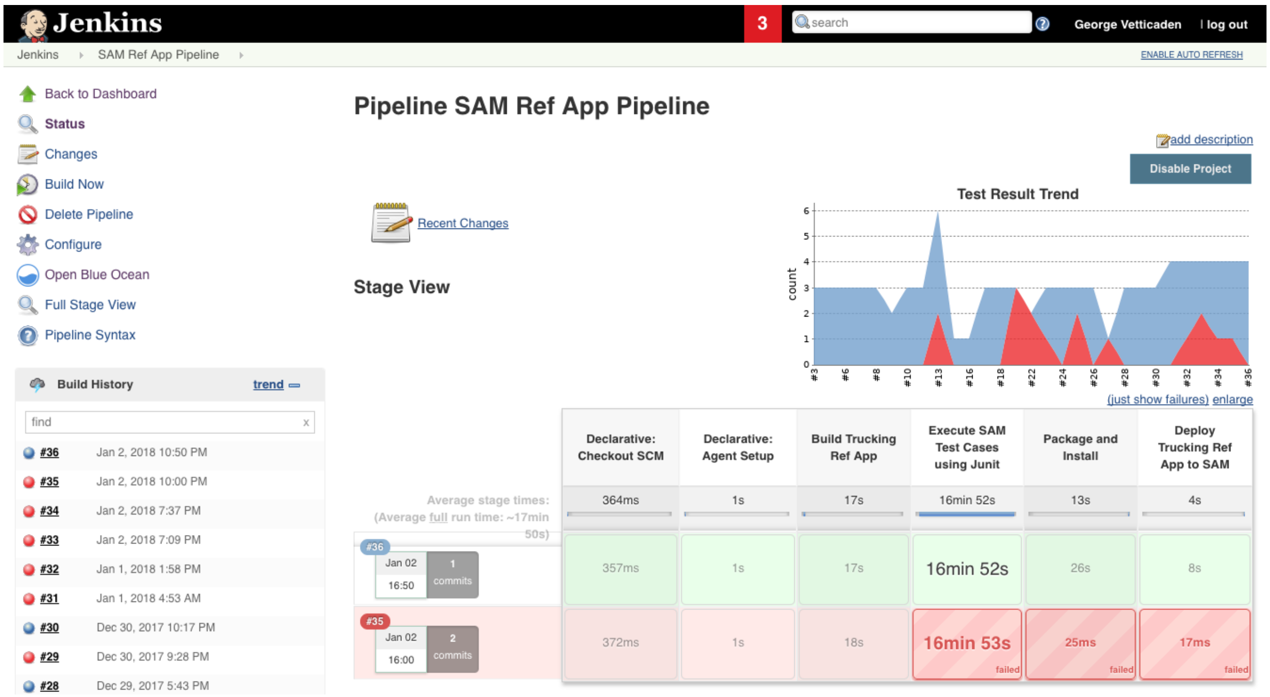
Each of these test cases will do the following:

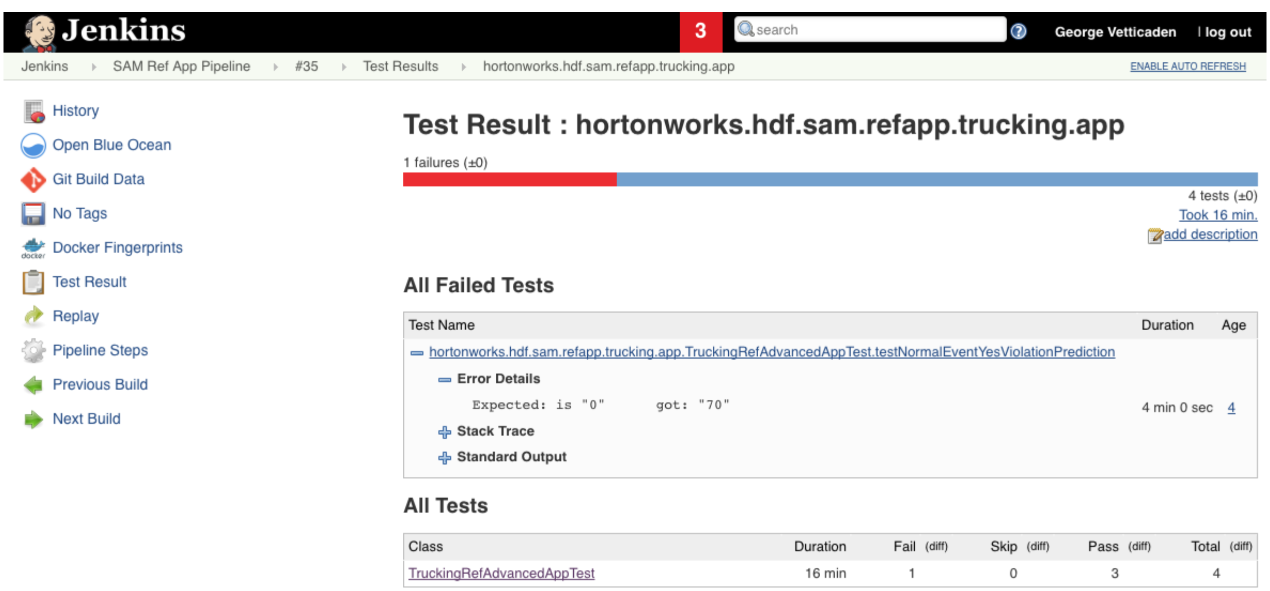
- Create the SAM Test Case
- Setup test data for each of the sources for each test case.
- Execute the SAM Test Case using SAM Test Mode and wait for test to complete.
- Download the results of the test case.

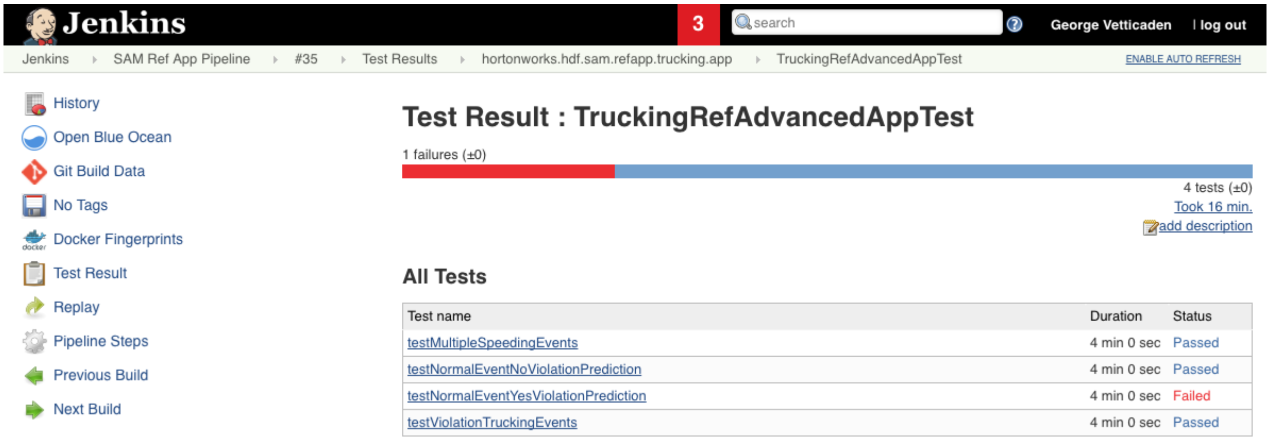
- Validate the results of the Test Case.

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









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

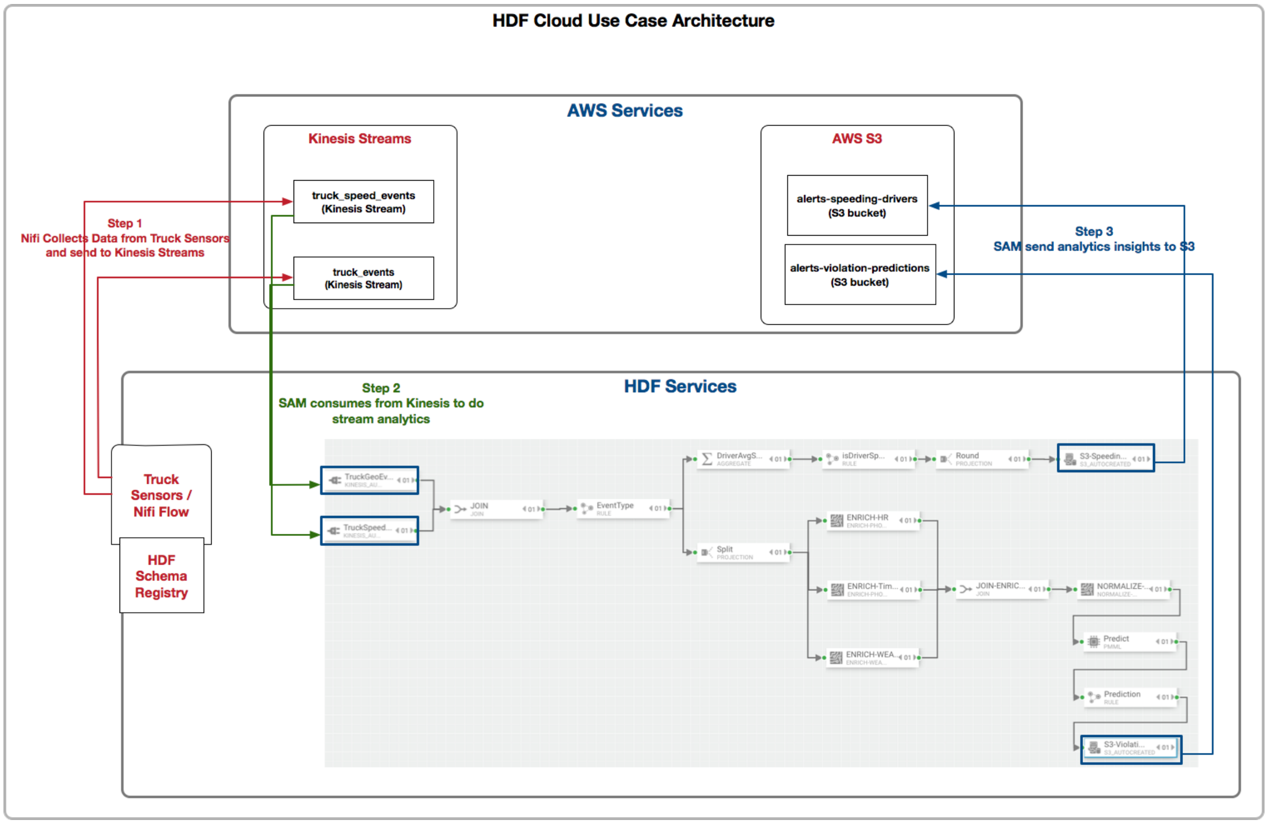
In this section, we walk through how to register custom sources and sinks in SAM integrated with Schema Registry.

Cloud Use Case: Integration with AWS Kinesis and S3

To showcase registering custom sources and sink, lets modify our Trucking Reference Application 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:



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
```

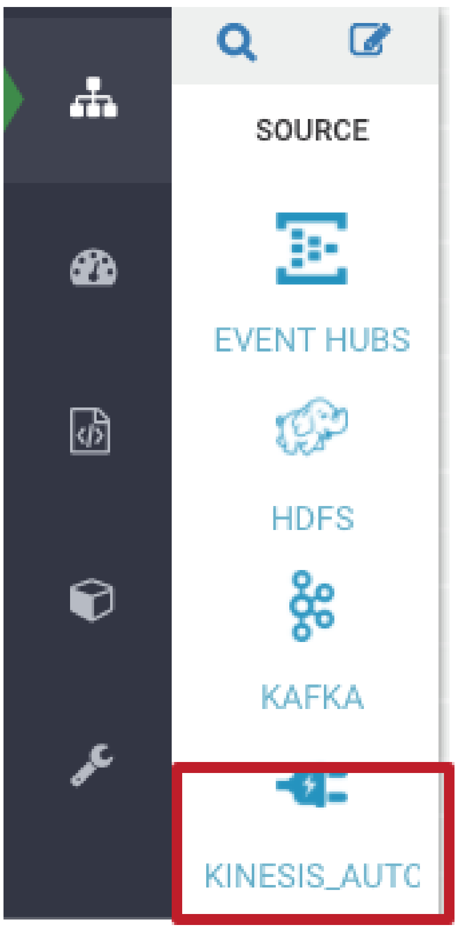
5. Register the custom source via SAM REST call. Replace SAM_HOST and SAM_PORT.

```
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
```

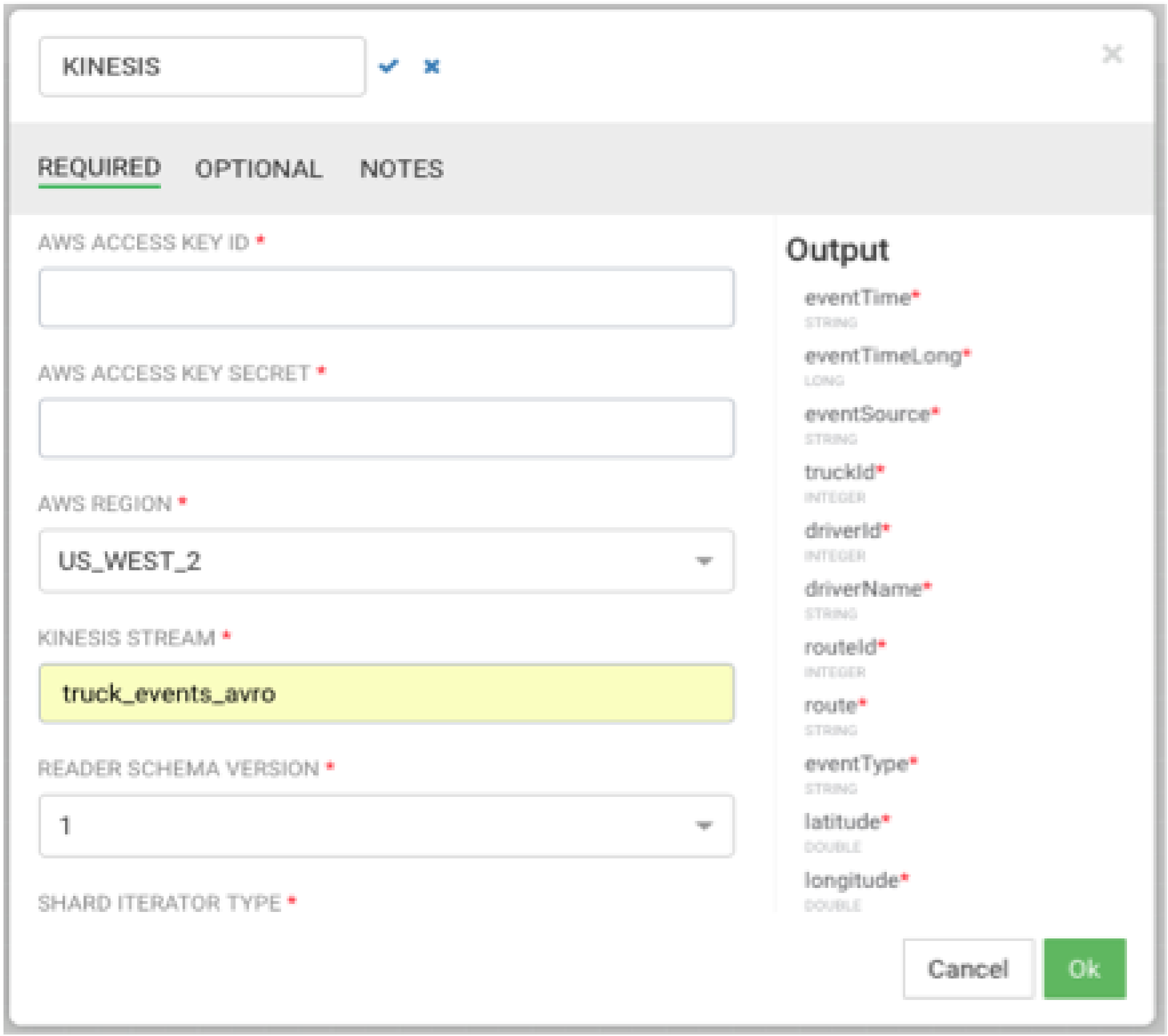
6. If the registration was successful, you should see a message like the following by the REST response:

```
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.



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.

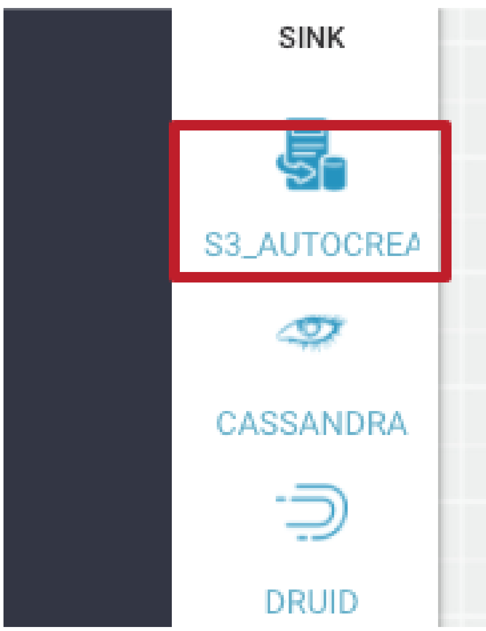
```
cd $SAM_EXTENSIONS/custom-sink/s3

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
```

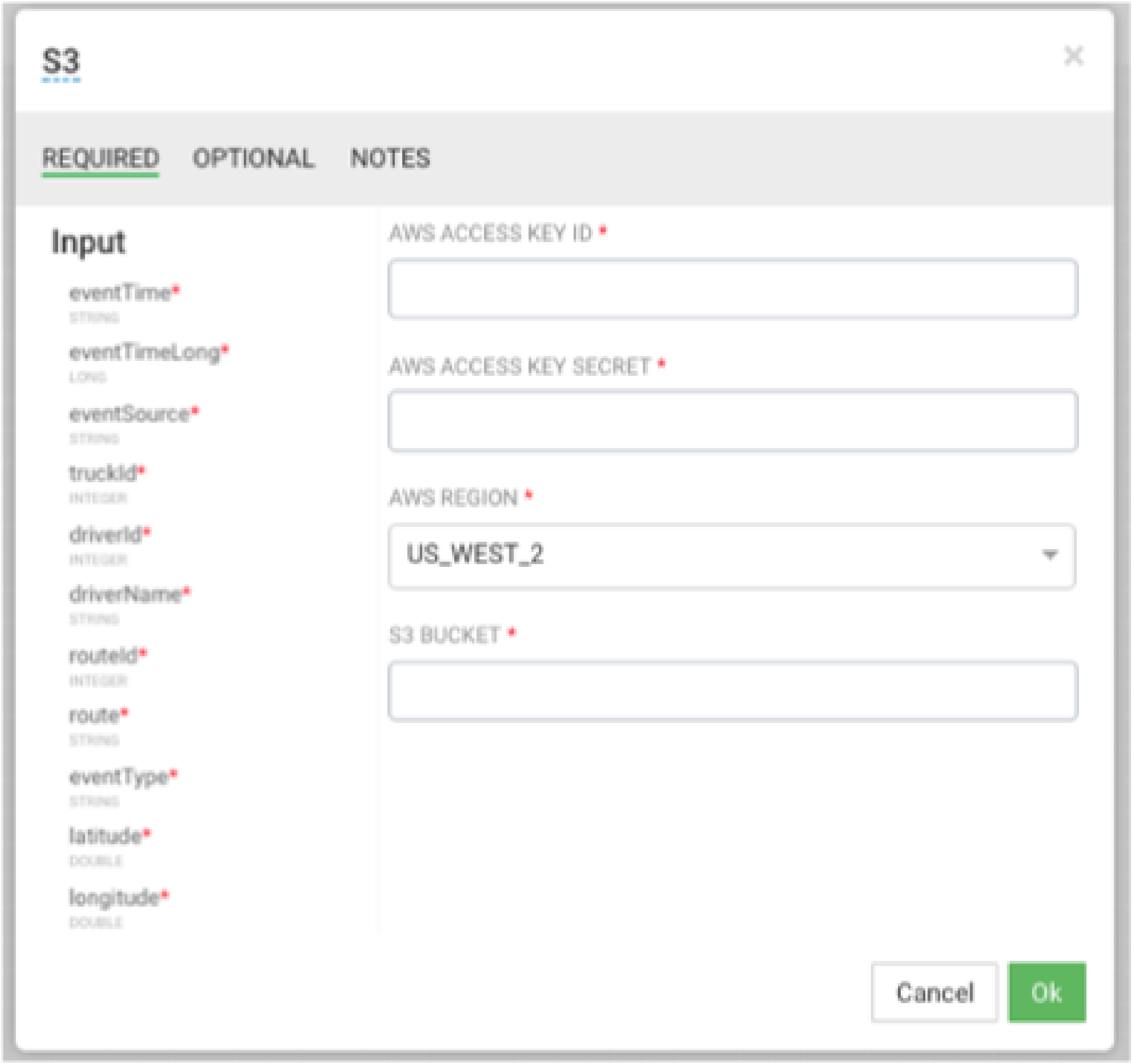
5. Register the custom sink via SAM REST call. Replace SAM_HOST and SAM_PORT.

```
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.

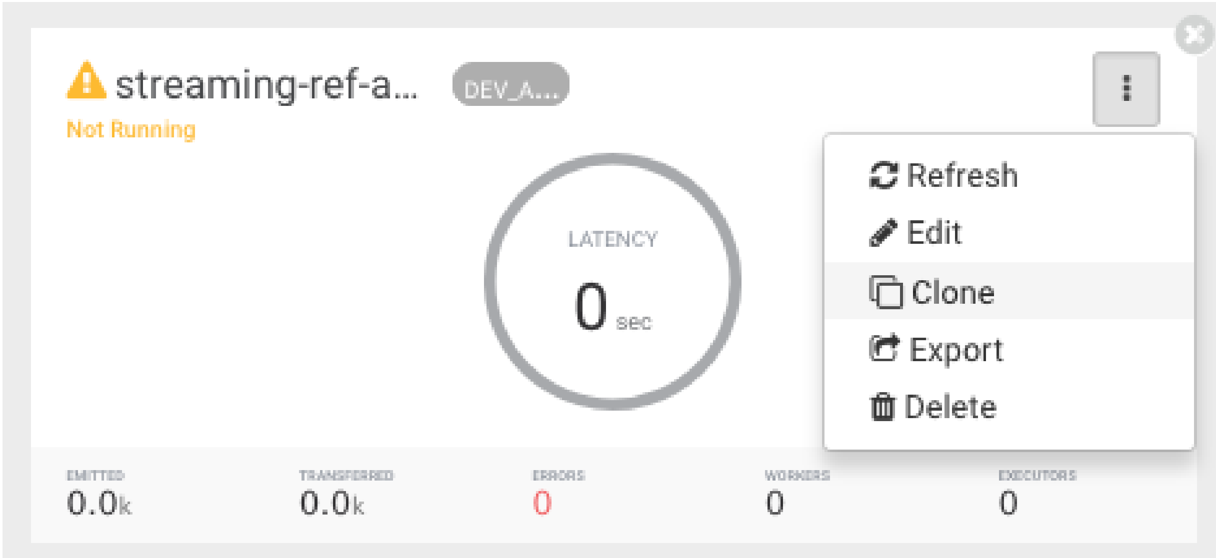


Implementing the SAM App with Kinesis Source and S3 Sink

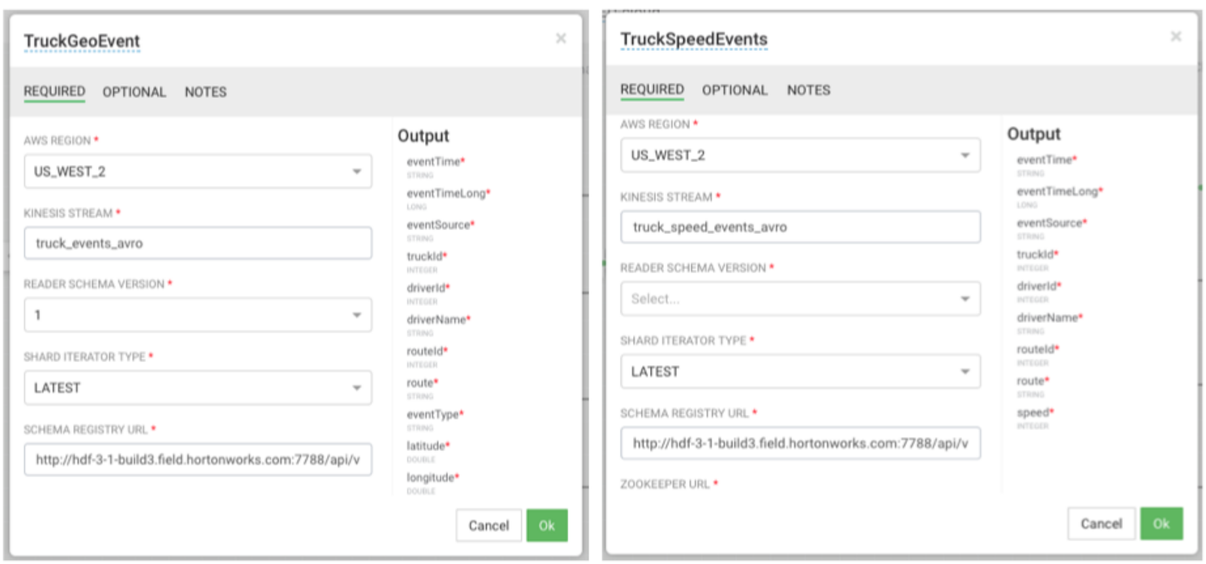
Now that we have registered teh custom Kinesis and S3 sources and sink, we can now build the streaming application in SAM to implement the cloud use case requirements.

Procedure

1. Clone the trucking reference application.



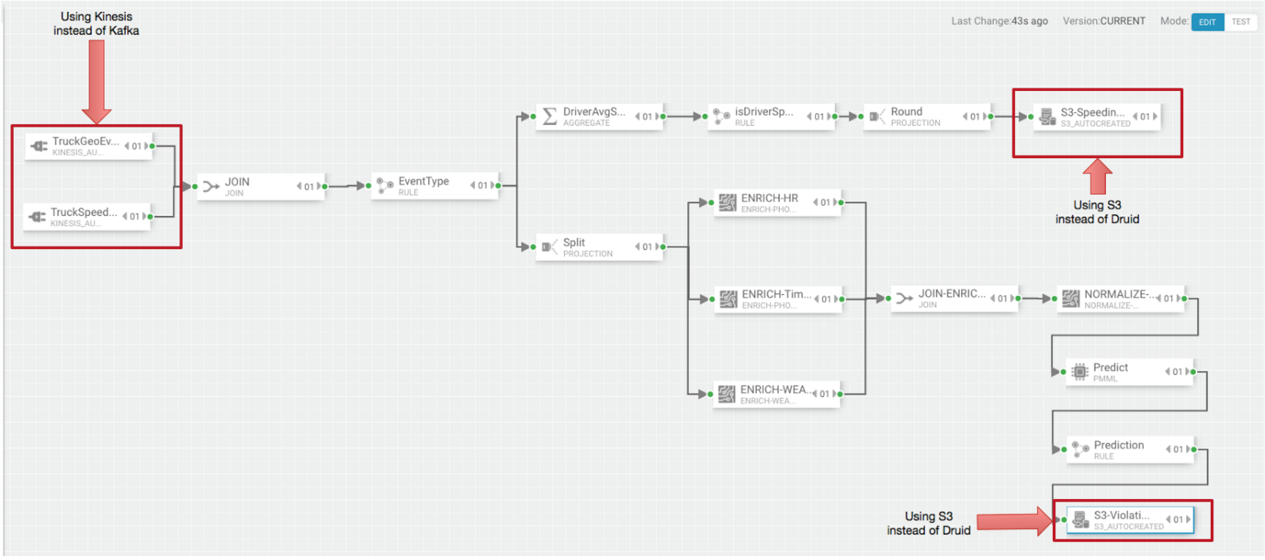
2. Rename the clone app to streaming-ref-app-advanced-cloud
3. Delete the Kafka sources and druid sinks from the SAM App
4. Add Kinesis sources for the deleted the Kafka Topics. Make sure to create the create the kinesis streams in AWS with the same names as the schemas you defined SAM's SR. You you 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.

Results

The SAM App should look like the following:



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

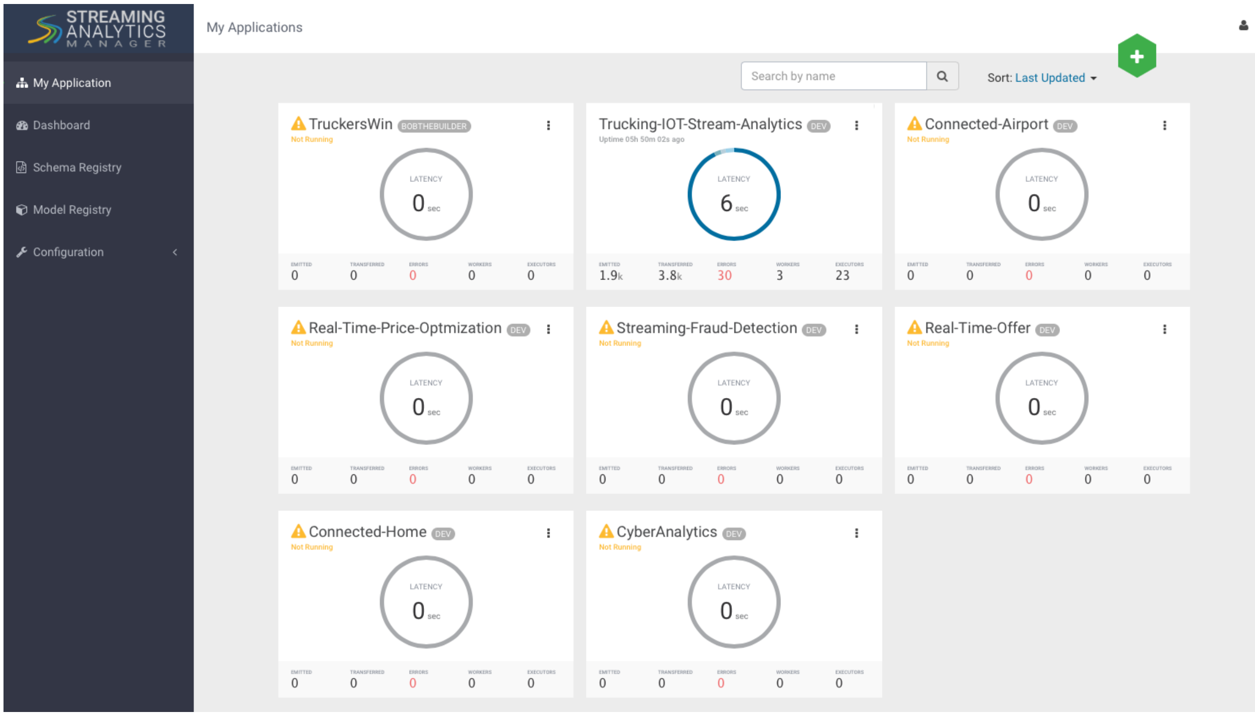
My Applications View

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.

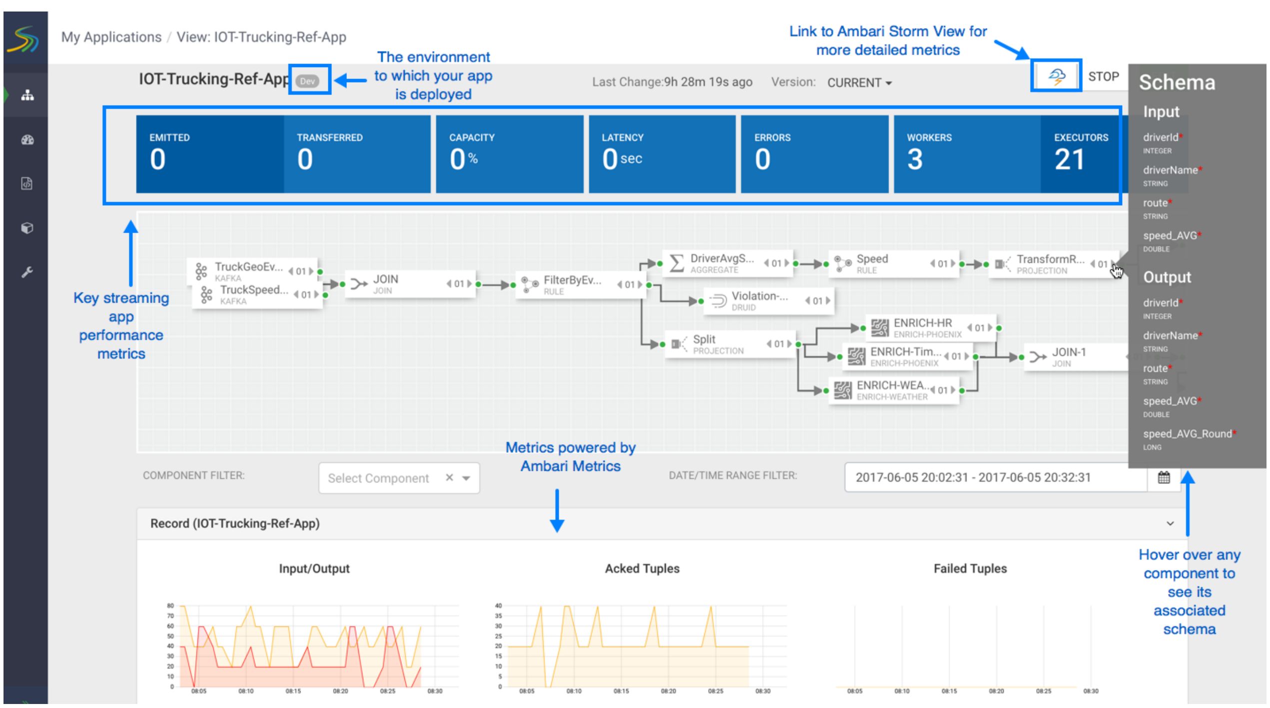
Each stream application is represented by an application tile. Hovering over the application tile displays status, metrics, and actions you can perform on the stream application.



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.

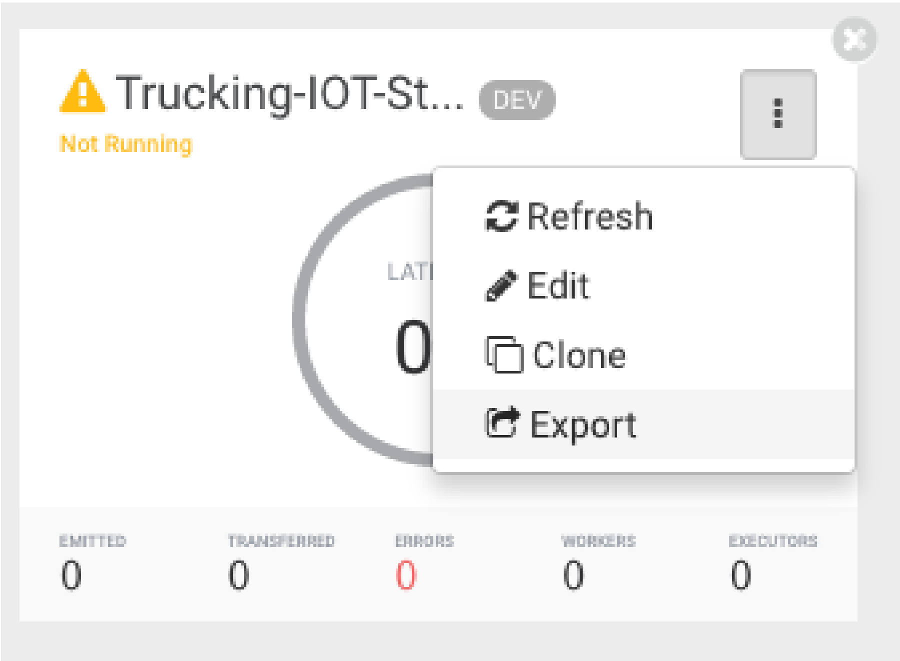


Exporting and Importing Stream Applications

Service pool and environment abstractions combined with import and export capabilities allow you to move a stream application from one environment to another. This task provides instructions about importing a stream application that was exported in JSON format.

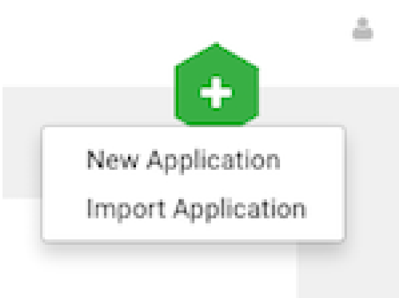
About this task

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

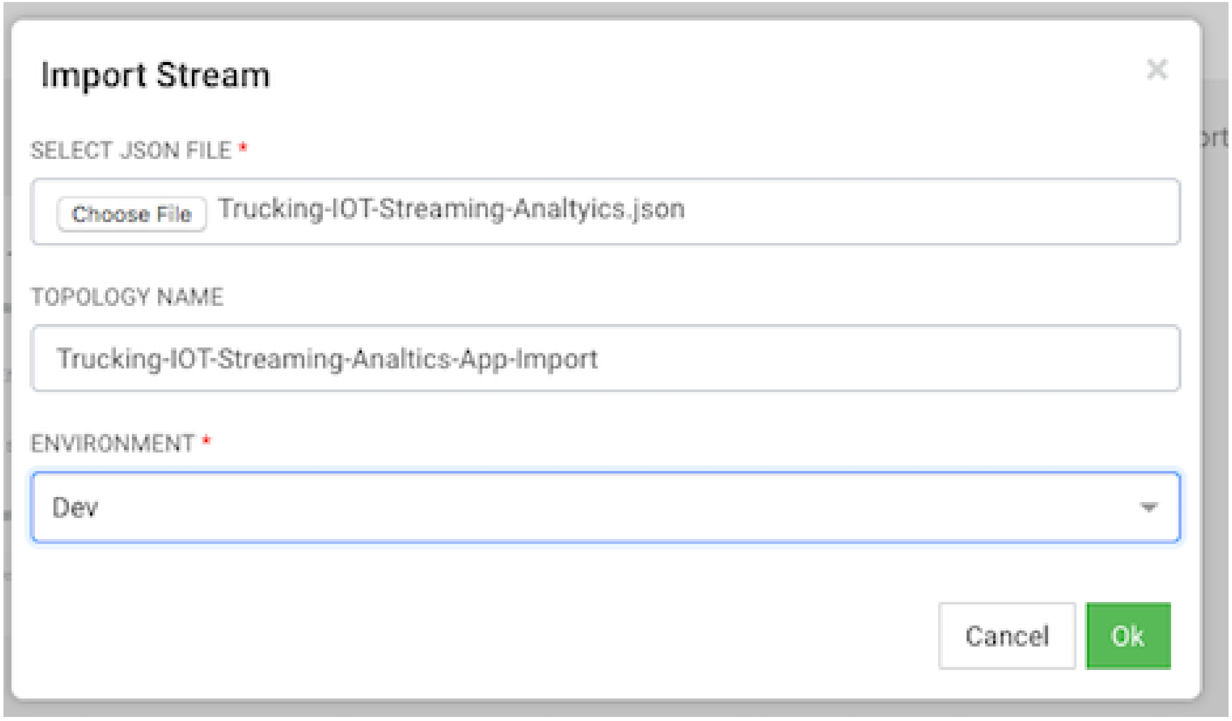


Procedure

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.



Troubleshooting and Debugging a Stream Application

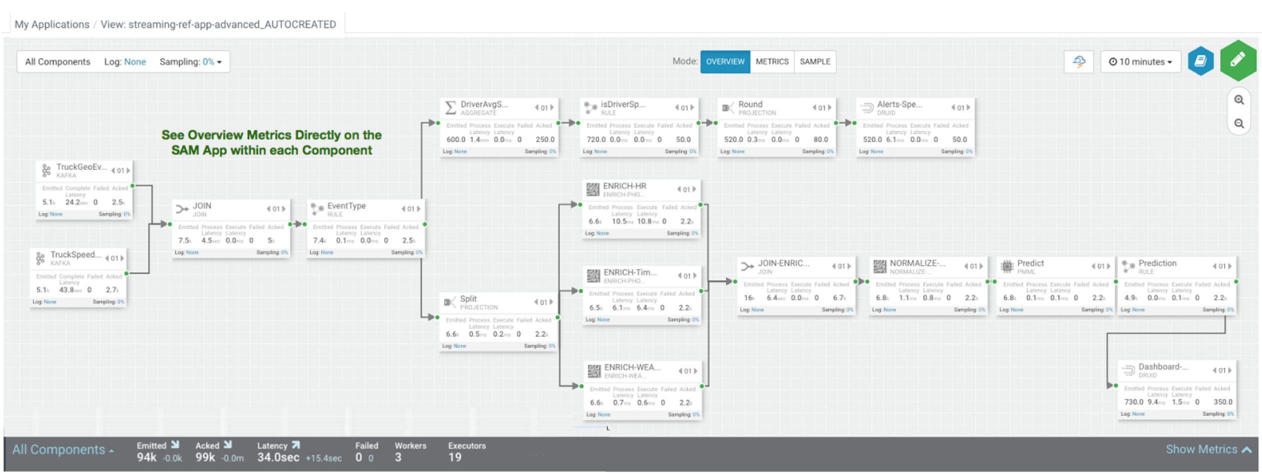
Once we have deployed the streaming app, common actions performed by users such as DevOps, Developers, and Operations teams are the following:

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

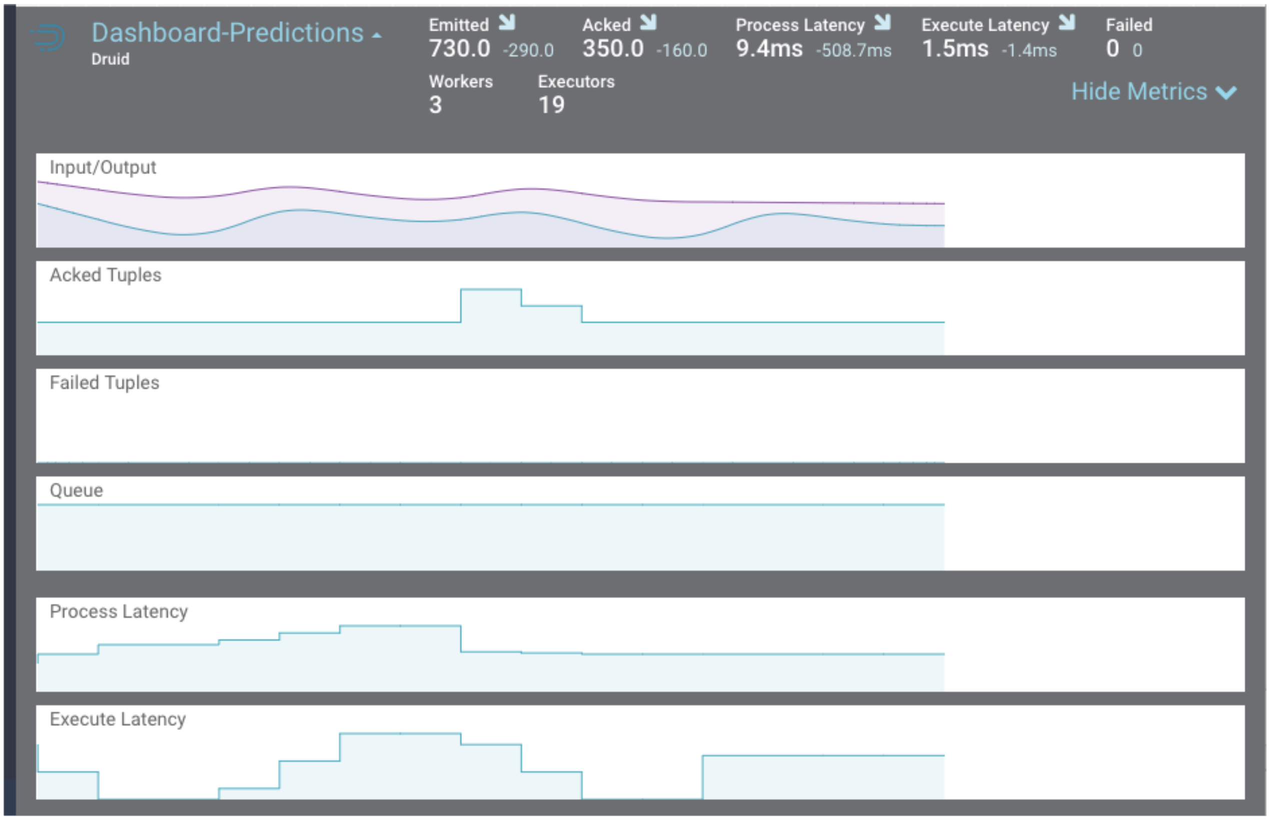
SAM makes performing these tasks easier by using the same visual approach as users have when developing the application. We will walk through these common use cases in the below sections.

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.



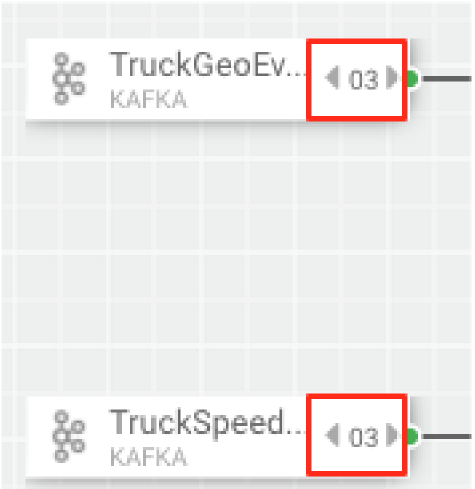
Key metrics include the following:

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.

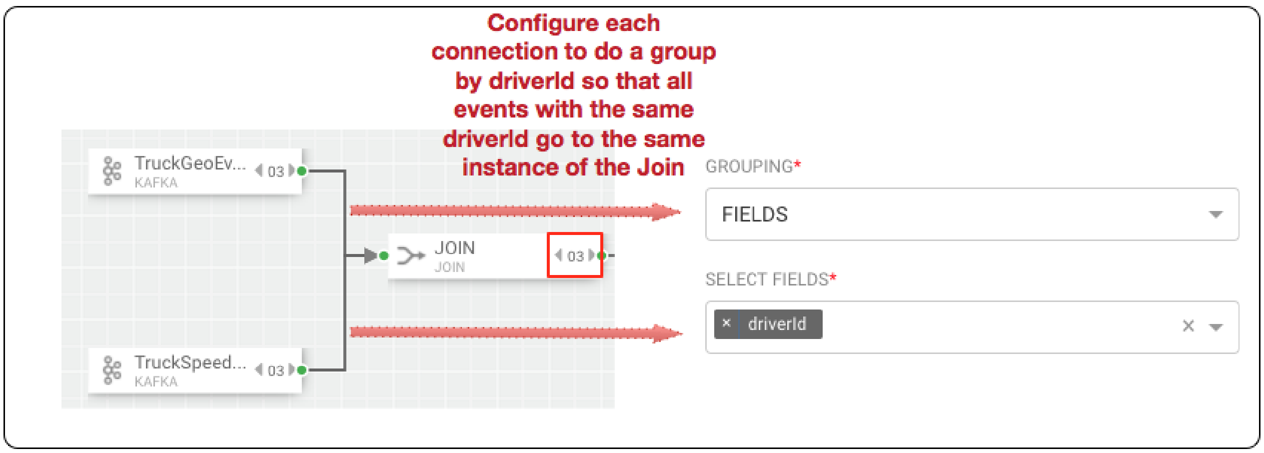
Identifying Throughput Bottlenecks

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.

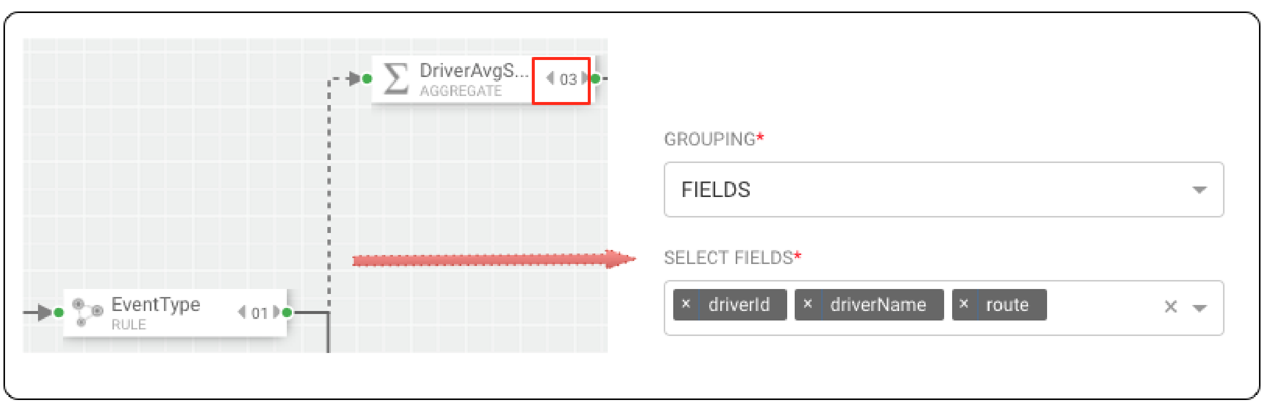
Increase the parallelism of TruckGeoEvent (kafka topic: truck_events_avro) and TruckSpeedEvent (kafka topic: truck_speed_events_avro) from 1 to 3. Note that each of these kafka topics have three partitions.



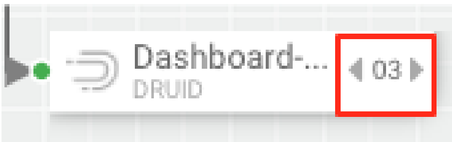
Increase the parallelism of the Join from 1 to 3. Since the join is grouped by driverId, we can configure the connection to use fields grouping to send all events with driverId to the same instance of the Join.



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.

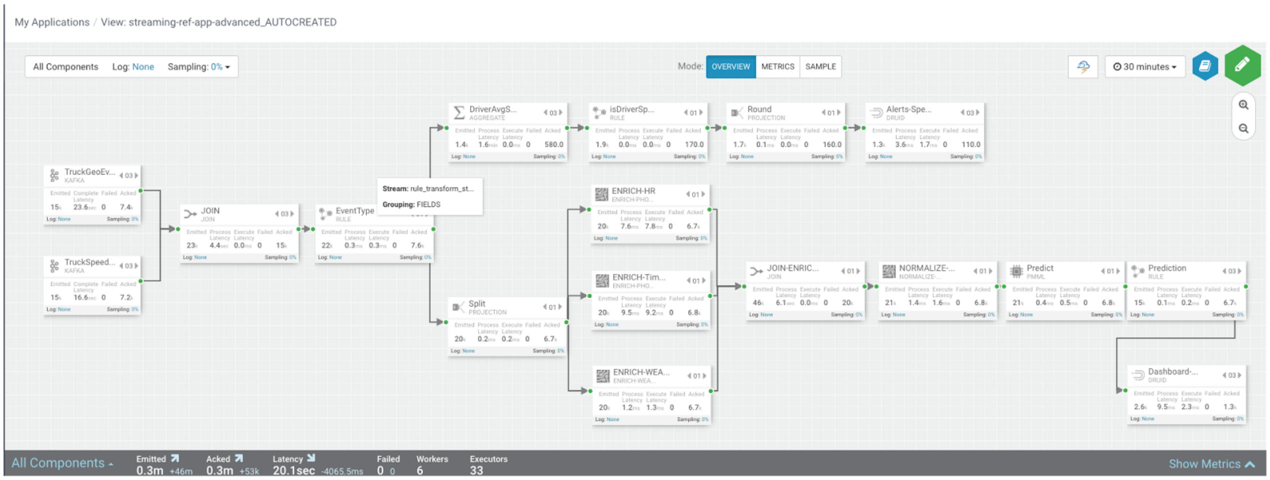


Increase the parallelism of the Dashboard-Predictions Druid sink from 1 to 3 so we can have multiple JVM instances of Druid writing to the cube.



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.

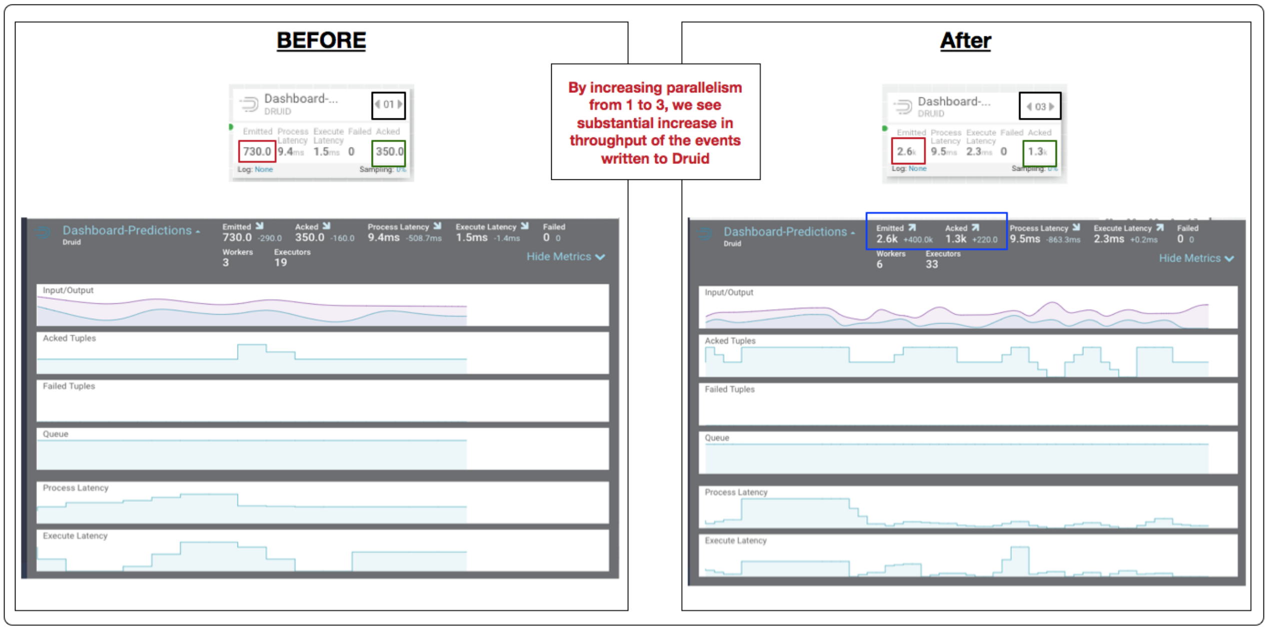
SAM's overview and detailed metrics makes it very easy to verify if the performance changes we made had the desired effect.



Throughput Improvements for the Kafka Source

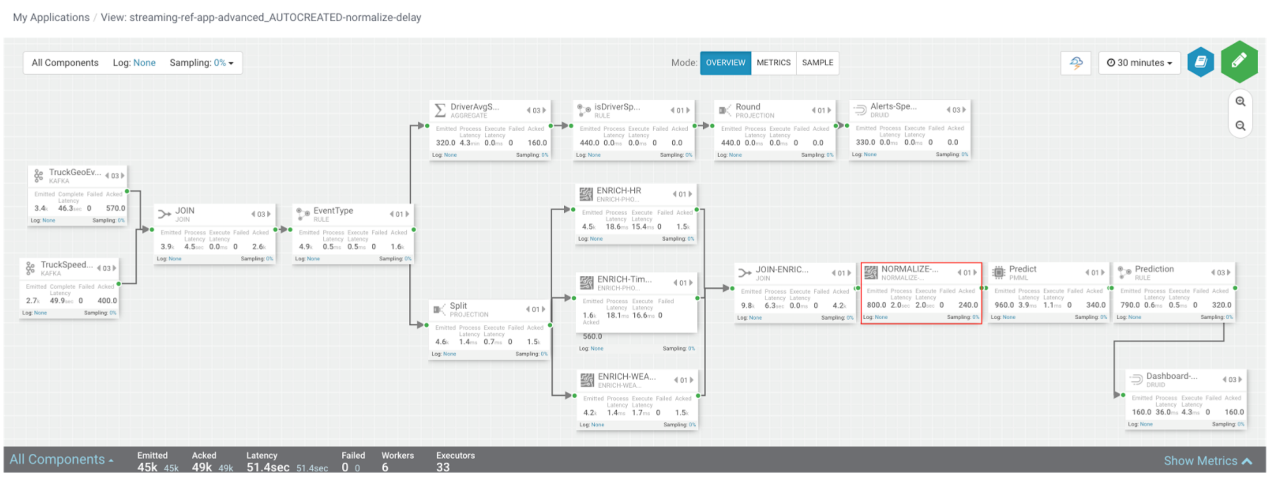
The below is the before and after metrics for the TruckGeoEvent Kafka Sink:

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

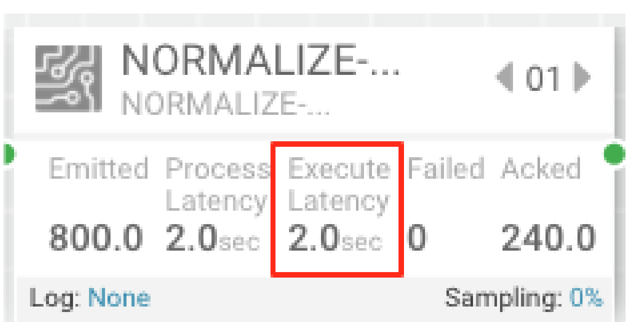


Identifying Processor Performance Bottlenecks

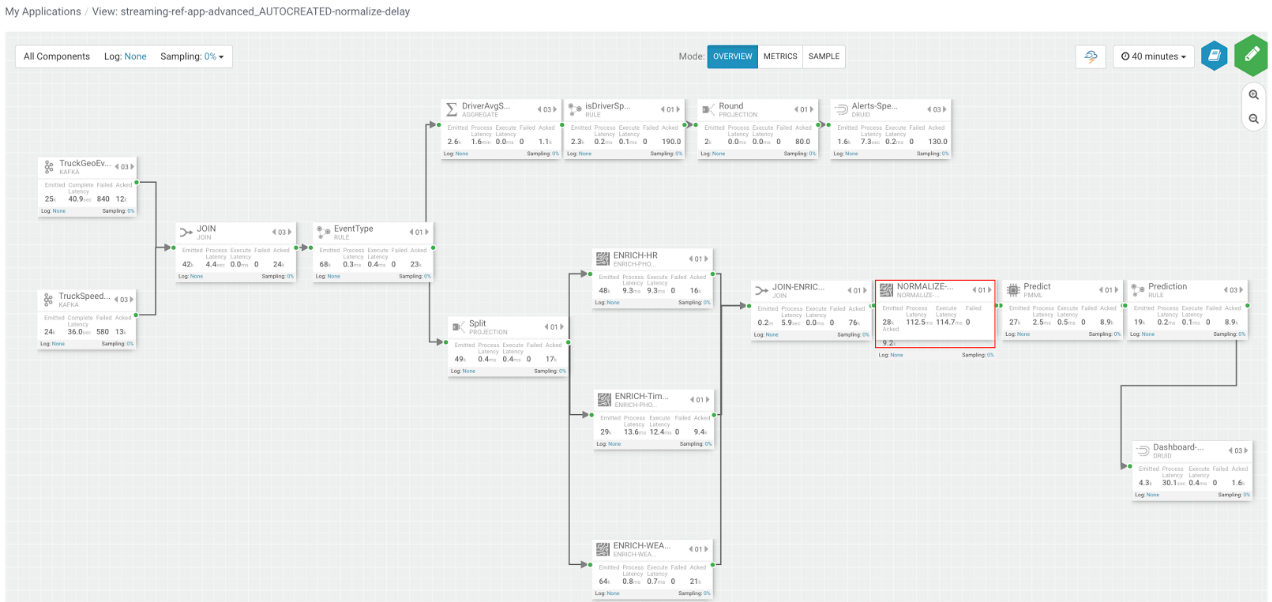
In this scenario, we identify a custom processor that has high latency. After running the data simulator for 30 mins, we view the Overview Metrics of the topology.



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.

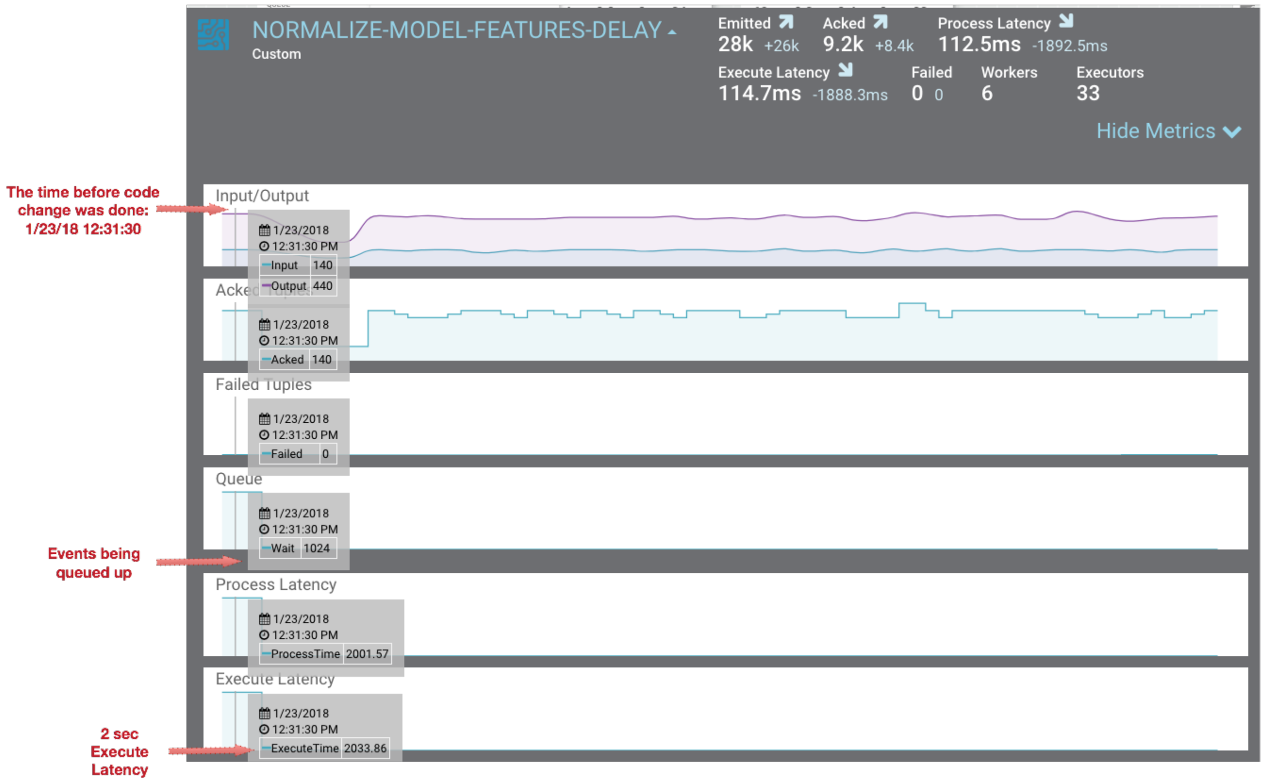


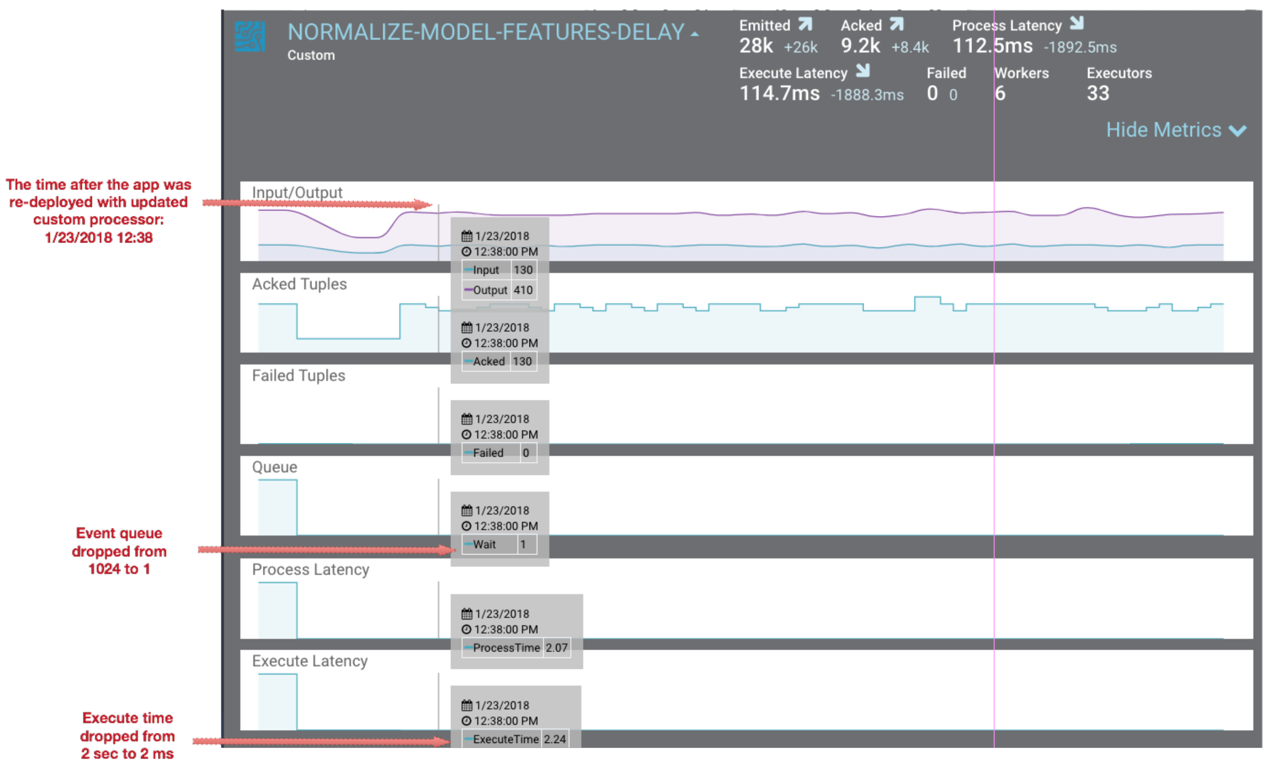
SAM's overview and detailed metrics makes it very easy to verify if the performance changes we made had the desired effect.

Latency Improvements

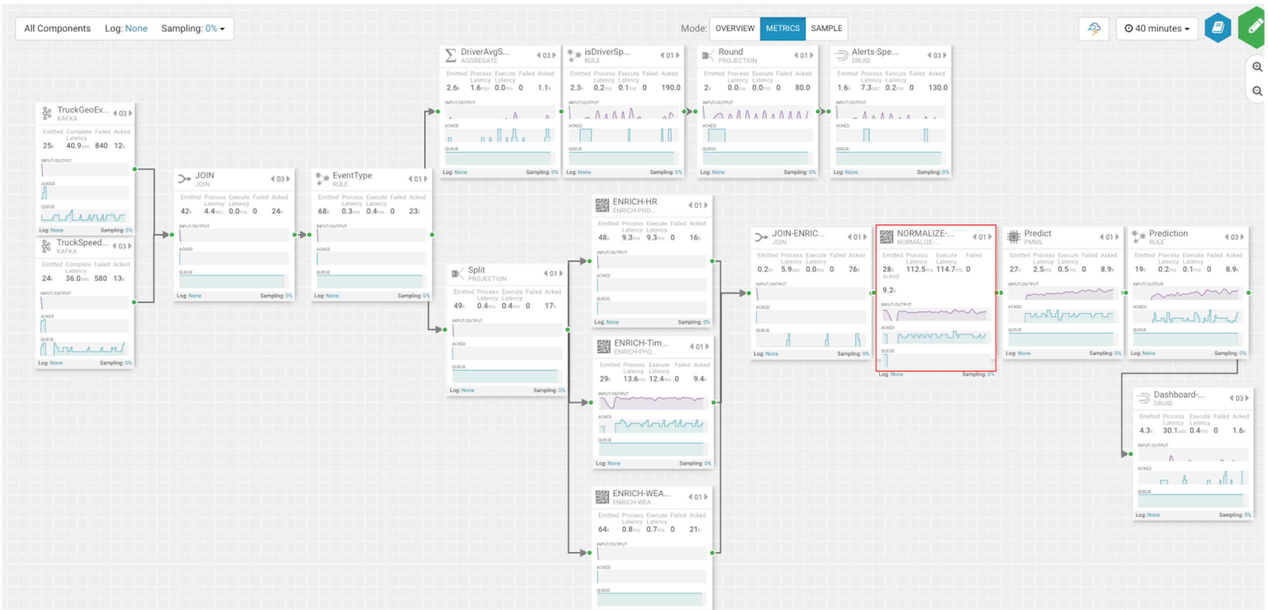
The below is the before and after metrics for the NORMALIZE-MODEL-FEATURES custom processor.

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

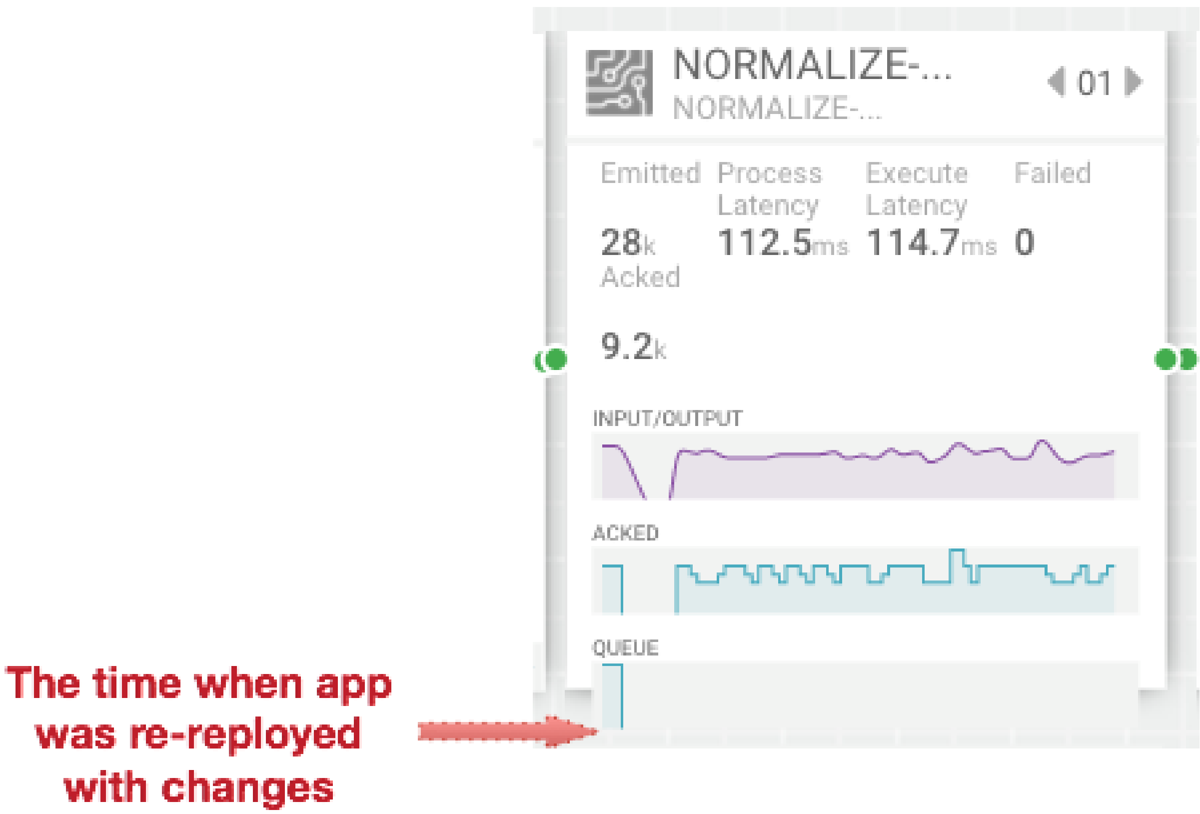




You can also select the Metrics tab to validate the performance improvement.



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.



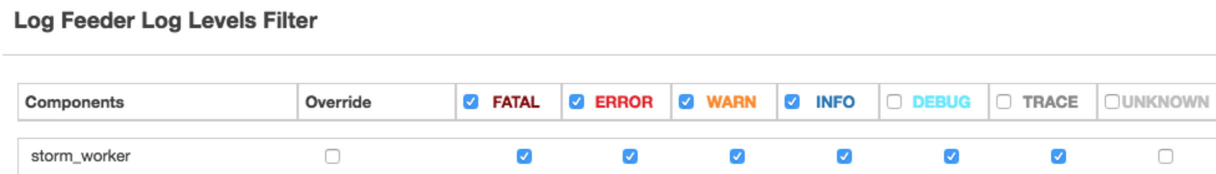
Debugging an Application through Distributed Log Search

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:

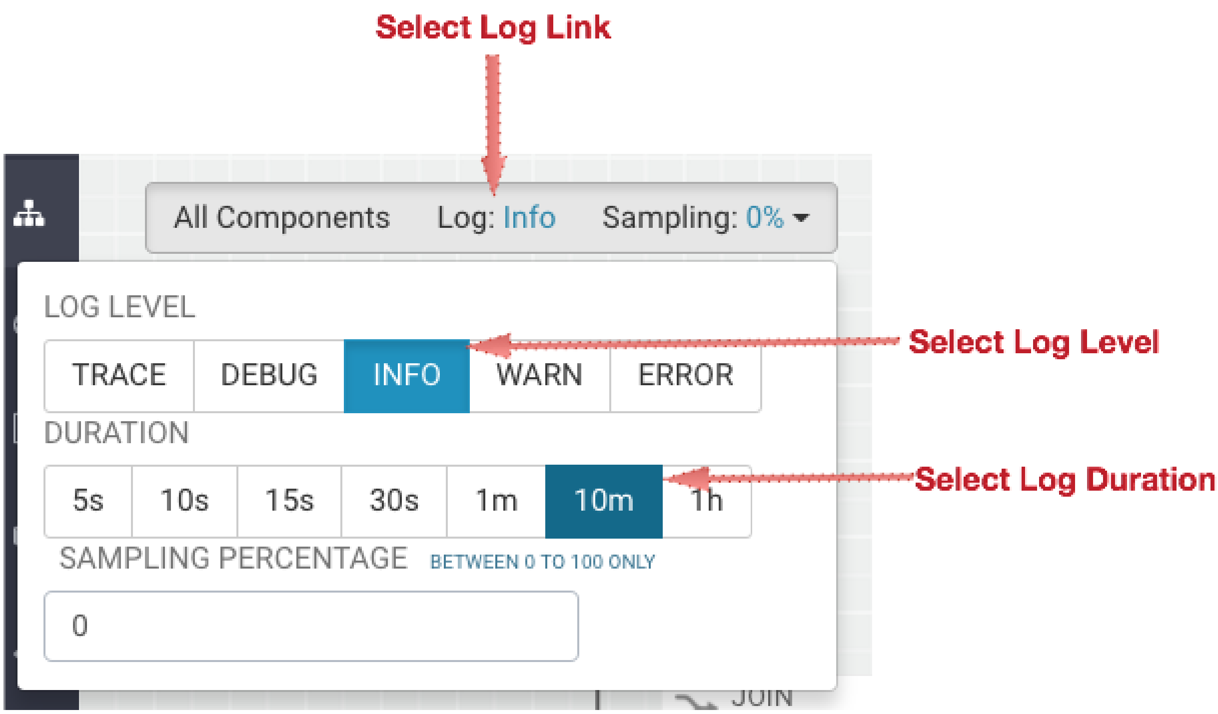
Procedure

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

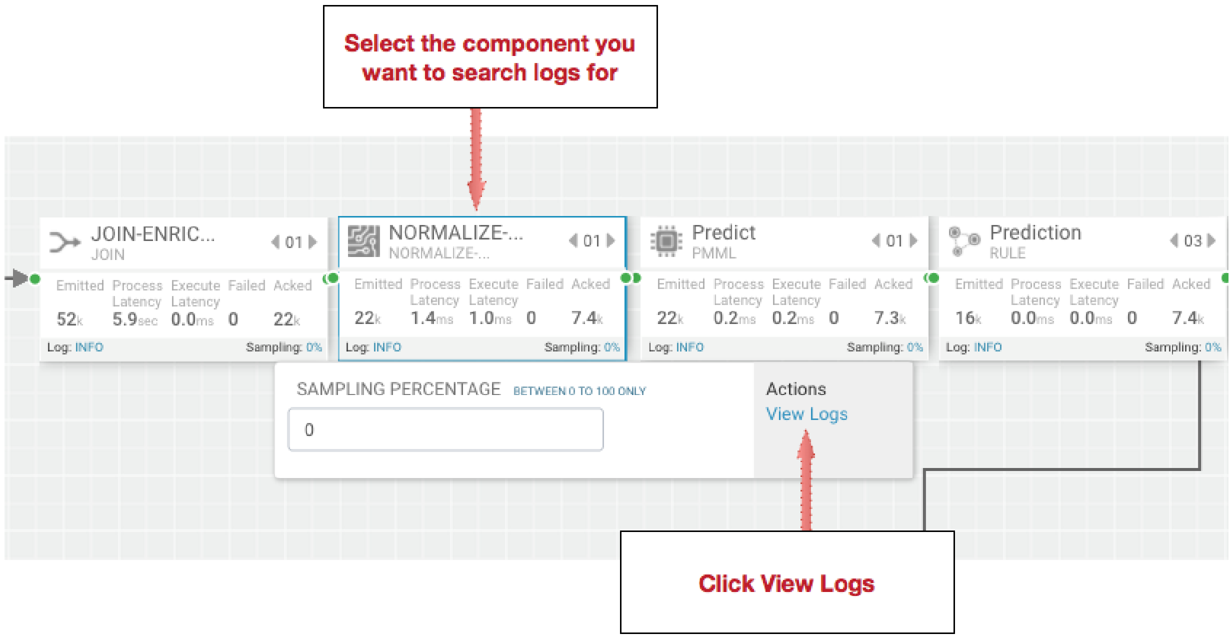
   a. In Ambari, select the Log Search service and select ‘Log Search UI’ from Quick Links.
   b. Select the filter icon on the top right menu.
   c. For the storm_worker component, configure the filter like the following and click Save.



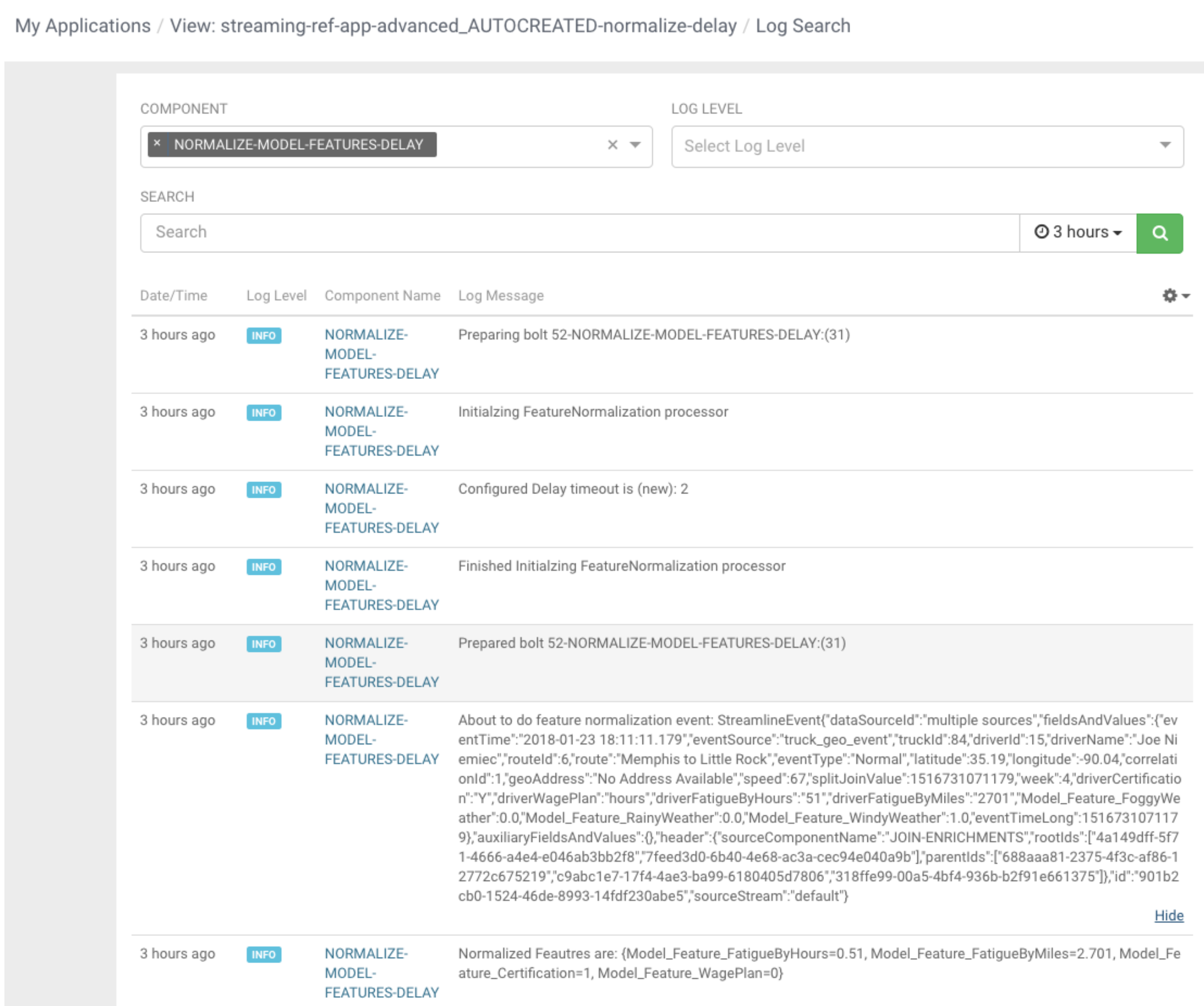
2. In SAM, you can dynamically change the logging level. For example, in SAM view mode of an application, click on the Log link, select the log level and the duration you want that log level.



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.

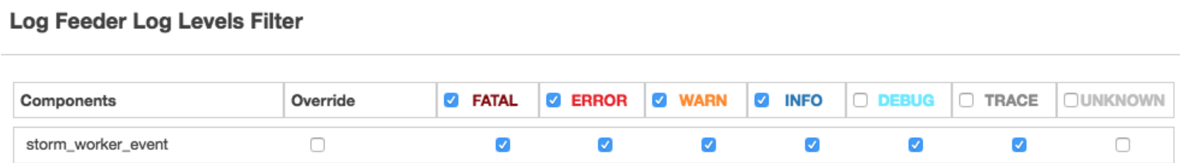


Debugging an Application through Sampling

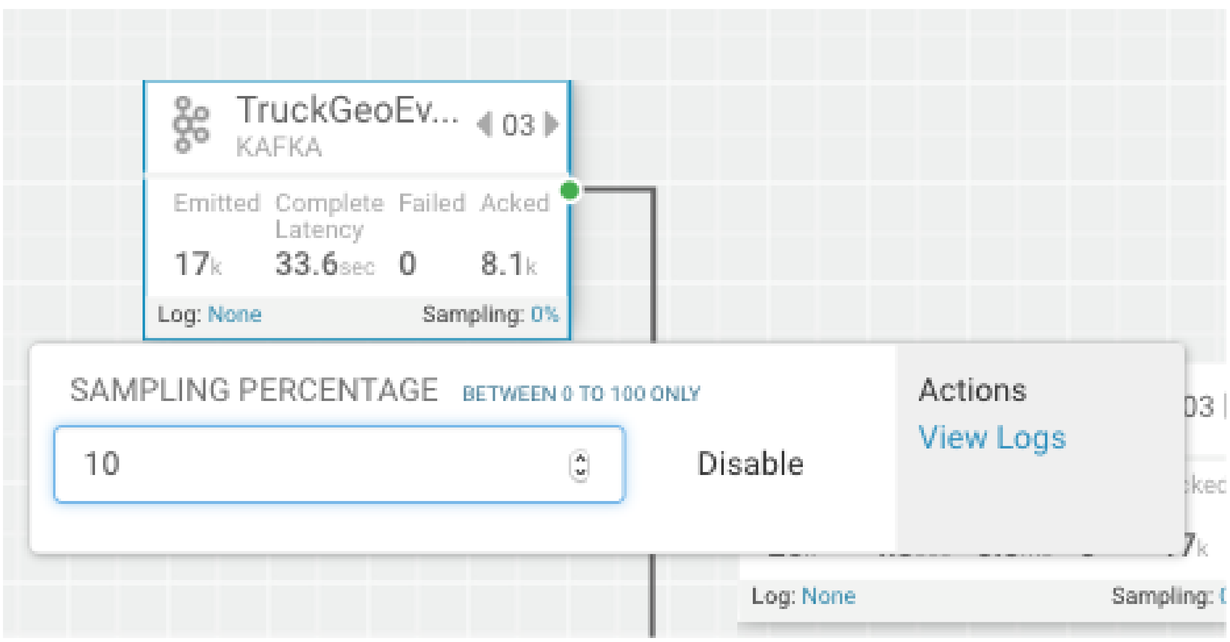
For troubleshooting, a convenient tool is to turn on sampling for a component or for the entire topology based on a sample percentage. Sampling allows you to log the emitted values of a component in the SAM App.

Procedure

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.

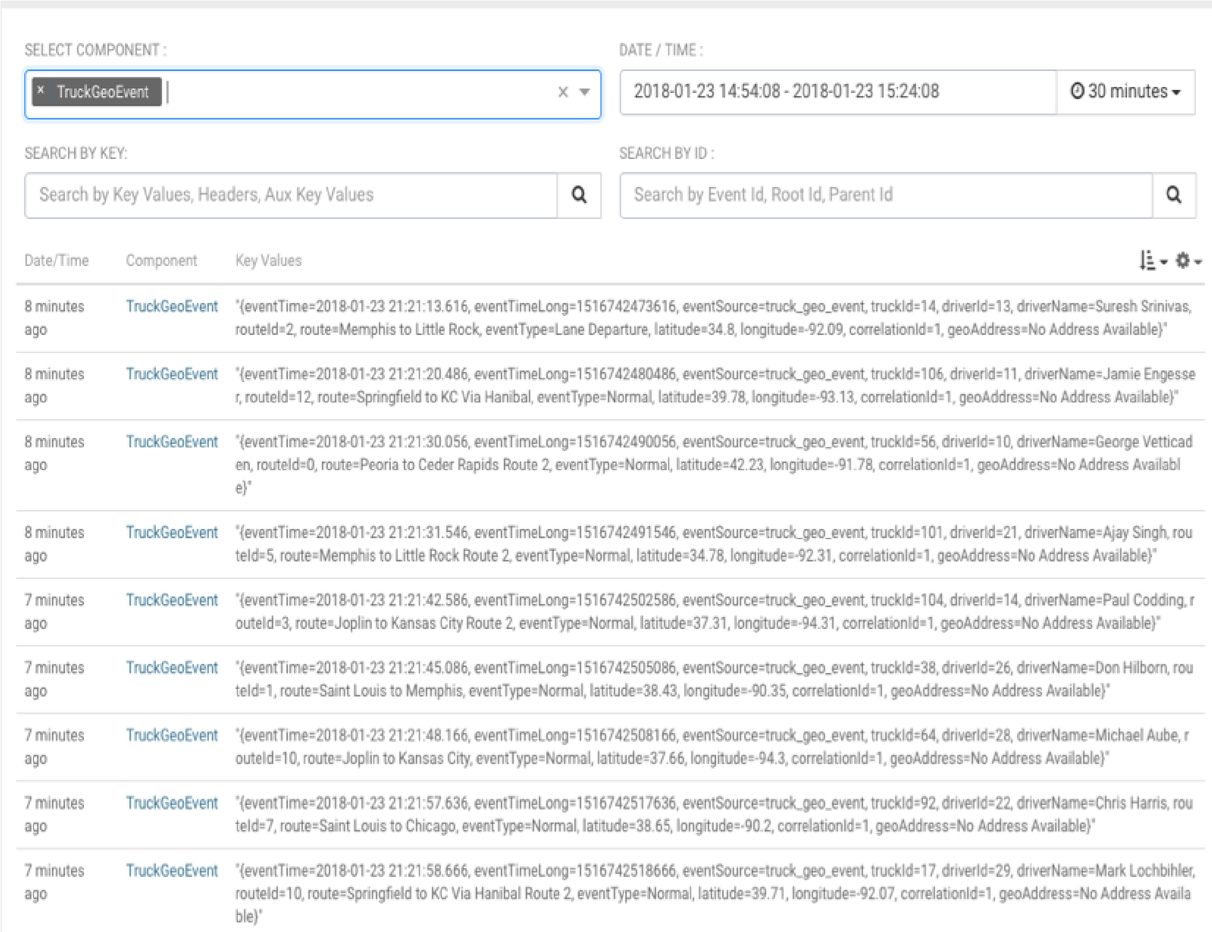


2. In SAM view mode of the App, click on the component you want to turn on sampling for and enter a sampling percentage.



3. Click the 'SAMPLE' Tab .

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



Spark Streaming

Information on how to create a Streaming Analytics Application with Spark Streaming is under development and will be available after the HDF 3.2.0 release is complete.

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.

About this task

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:

Procedure

1. Download the Data-Loader.
2. Unzip the Data Loader file and copy it to the cluster. Lets call the directory to which you unzipped the file as $DATA_LOADER_HOME.

3. Change into the Data Loader directory:

```
cd $DATA_LOADER_HOME
```

4. Untar the route.tar.gz file:

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

5. Open the file startTruckGenerators.sh and make the following changes:

   a. Modify the kafkaBrokers value based on your cluster
   b. If your cluster is not secure, set the value of SECURE_MODE to NONSECURE and set JAAS_CONFIG to an empty space
   c. Set the value of ROUTES_LOCATION to the location where you untar the routes in step 4 and then (e.g: $DATA_LOADER_HOME/routes/midwest)

6. Run the simulator/data generator:

```
./startTruckGenerators.sh
```

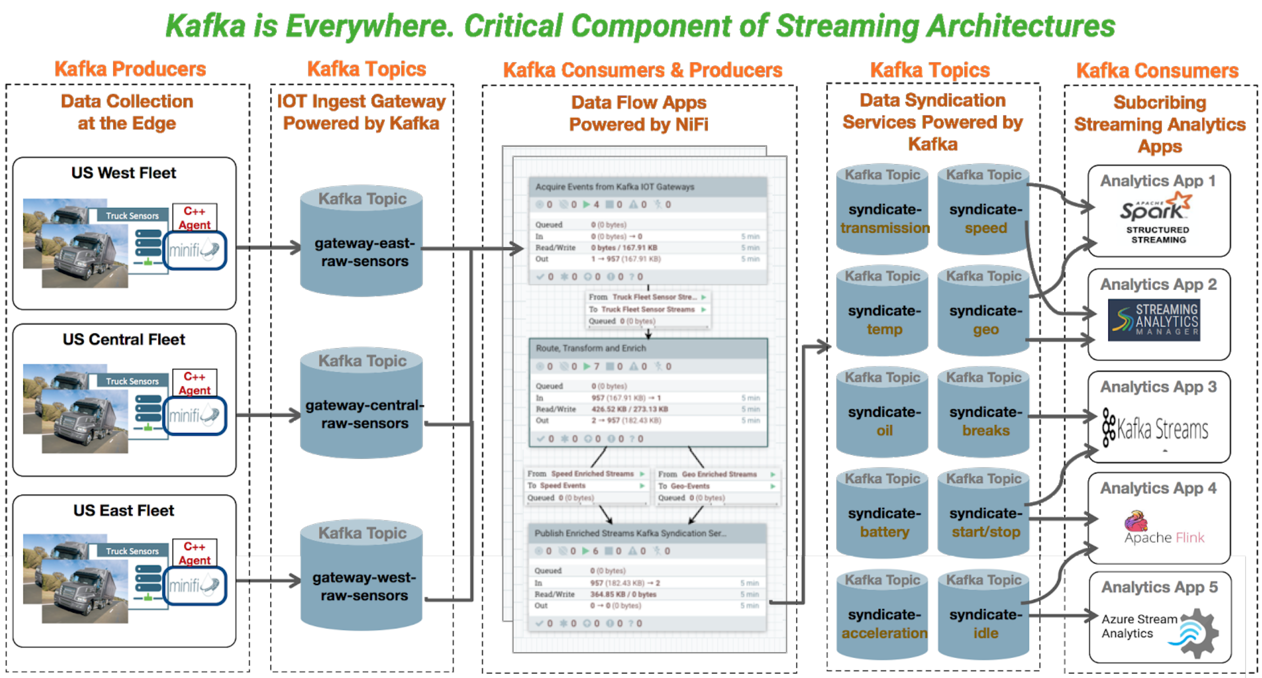
Results

You should see events being sent to the gateway kafka topics. NiFi should consume the events, enrich them, and then push them into the syndicate topics. SAM, Spark Streaming and other apps should consume from these syndicate topics

Managing Kafka with Streams Messaging Manager

SMM Overview

A key part of this streaming application has been the use of Kafka which powers the IOT Gateway and the Syndication Services. In this architecture, Kafka is everywhere.



As a result, it becomes critical to be able to monitor and understand what is going on in the cluster to cure the Kafka blindness. To accomplish this, we will use Hortonworks Streams Messaging Manager to monitor the Kafka components of this reference application.

Installing DataPlane Streams Messaging Manager

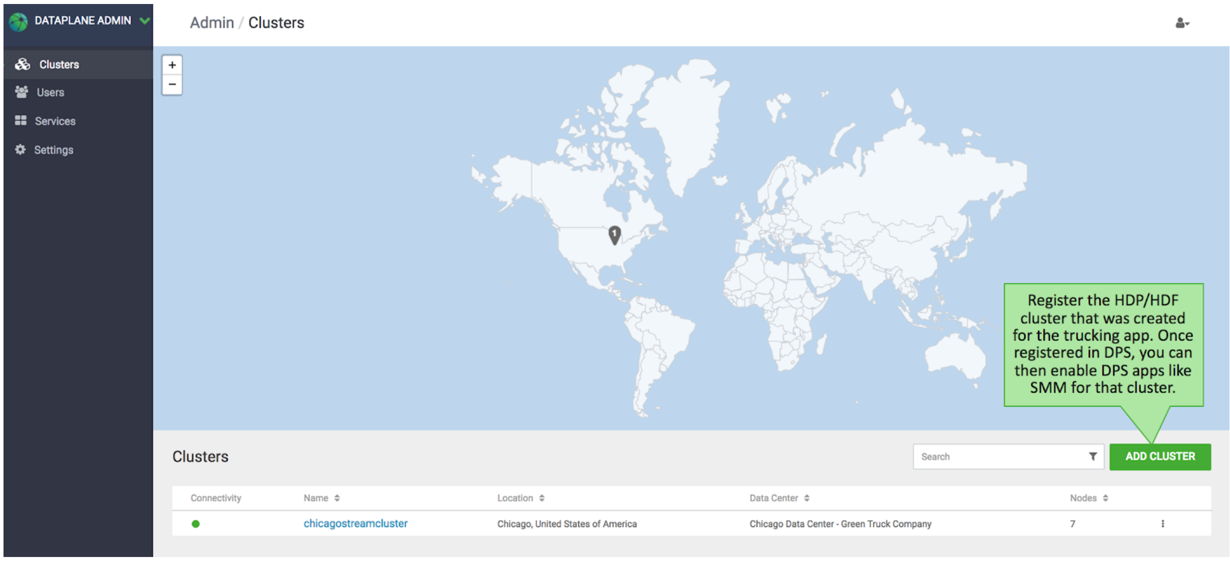
Follow the *SMM Installation* documentation to install SMM which requires DataPlane Service (DPS) platform as well as certain prerequisites required on the HDP/HDF cluster where Kafka is running.

Enabling Reference Application Cluster for SMM

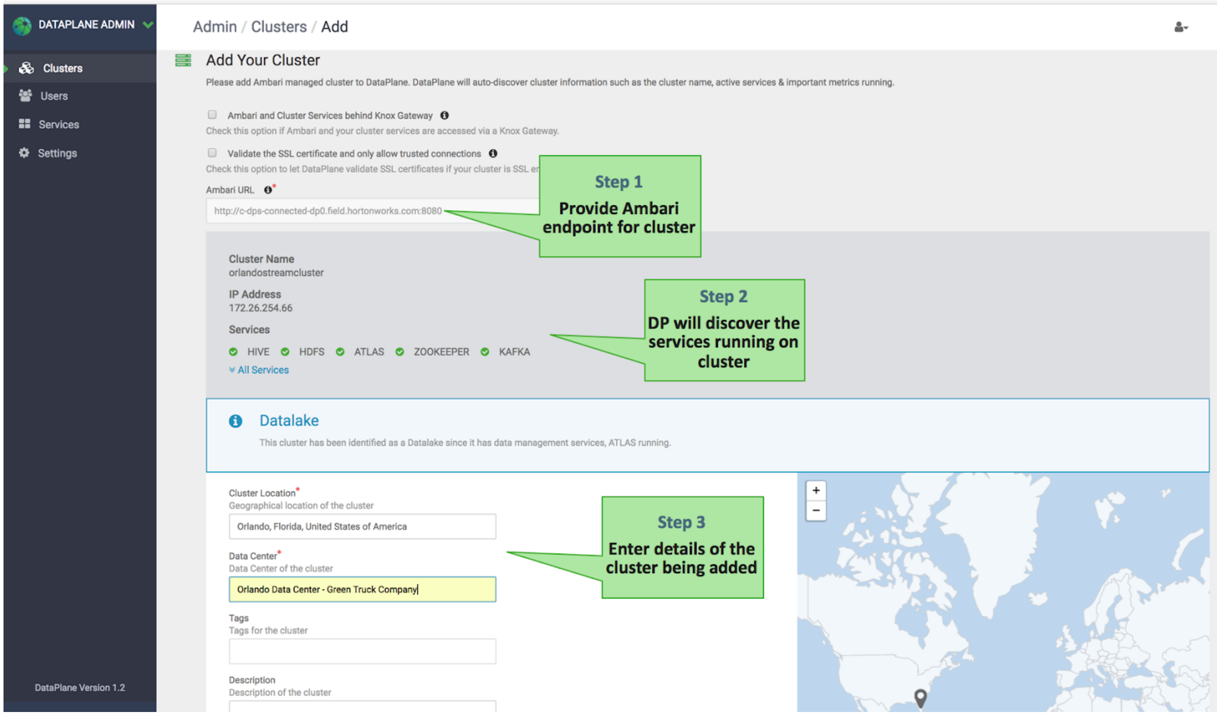
After installing DPS with the SMM application, you need to register the cluster you have created for the trucking reference application. The below steps walks you through this registration process.

Procedure

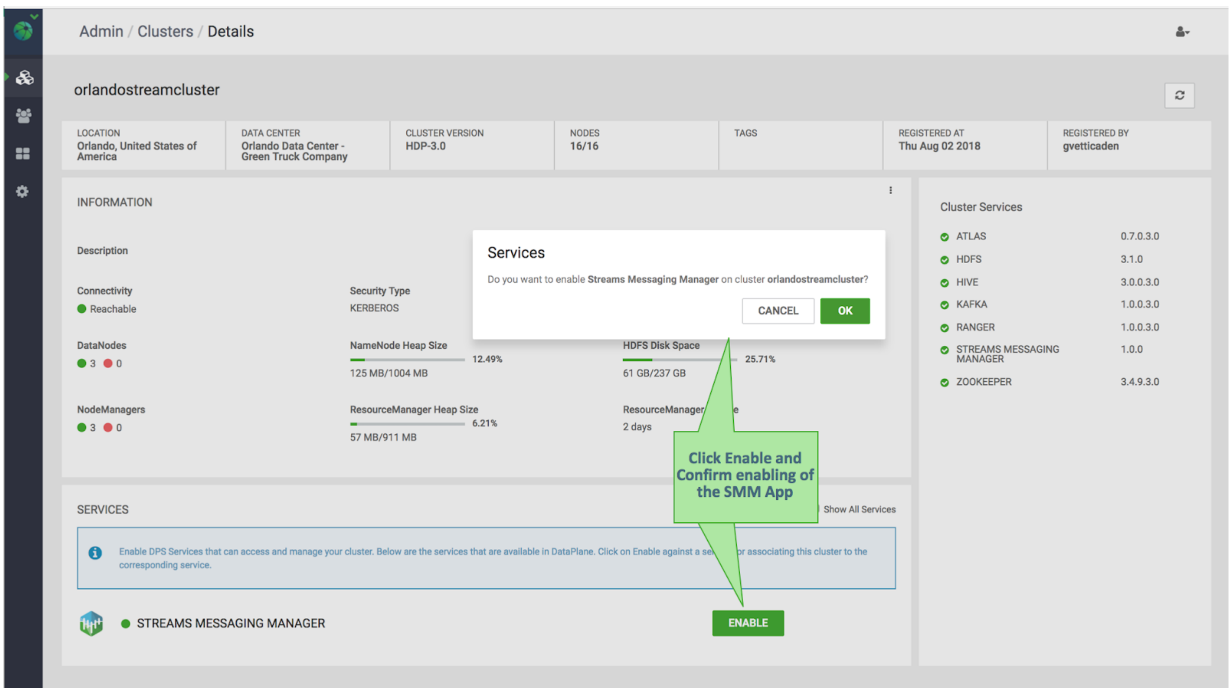
1. Log into DataPlane that was installed and click **Add Cluster**.



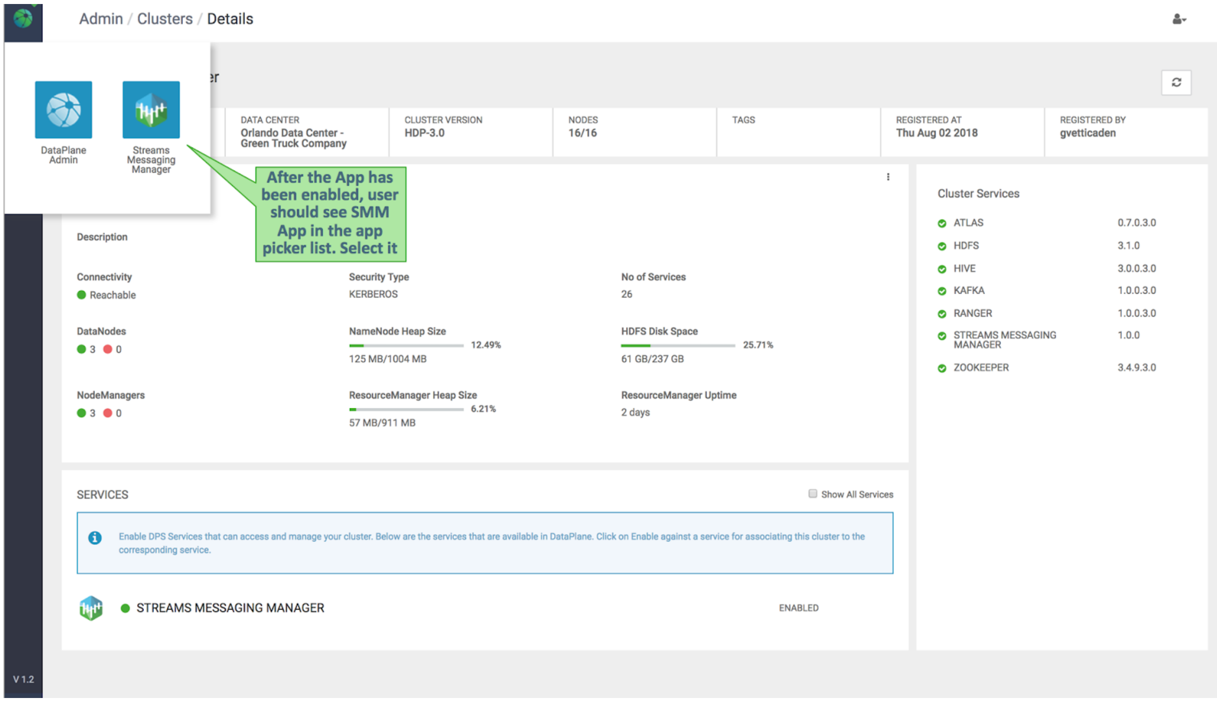
2. Provide Ambari endpoint and details for the HDP/HDF cluster that is being added to DataPlane.



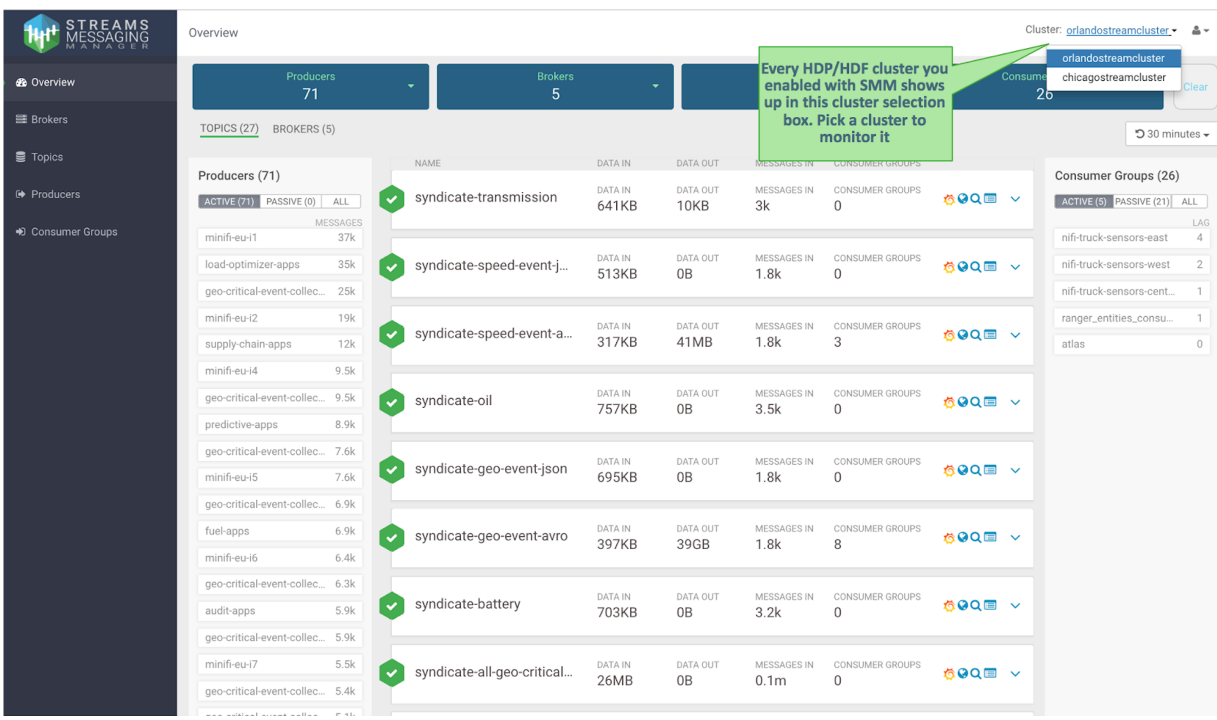
3. After the cluster has been added, go to the cluster details page and enable the SMM application.



4. Once the SMM app is enabled, you should see the SMM Icon from the app picker. Click on the SMM App to start monitoring the Kafka brokers in the cluster you registered.



5. In the SMM App, every HDP/HDF cluster you enabled with SMM shows up in the cluster dropdown. Hence, a single SMM App can monitor multiple clusters. Select the cluster you want to monitor.



Monitoring Kafka with SMM

SMM helps address the operational, management and monitoring needs of Kafka for two distinct teams: the Platform Operations and the DevOps / AppDev teams. Each of these teams have a different lens through which they monitor Kafka and hence have different needs for monitoring Kafka.

SMM Platform Operations Persona

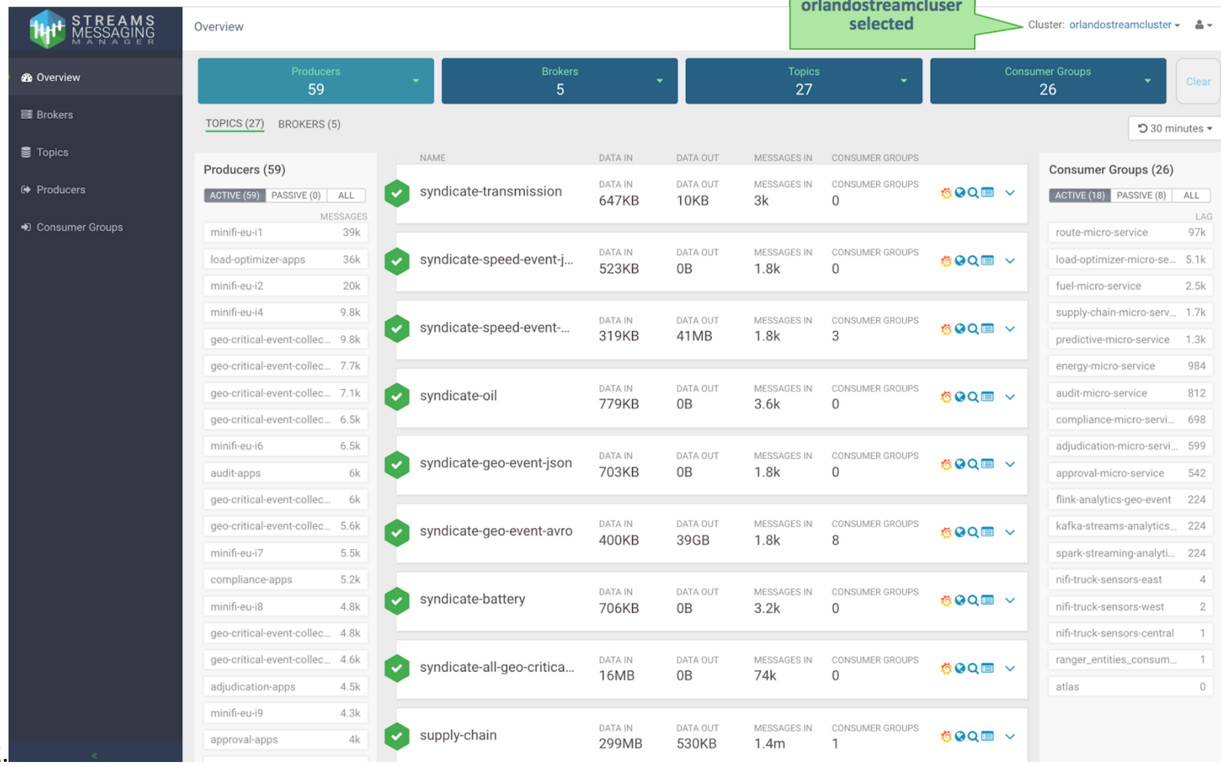
A Platform Operations user is less concerned about the individual performance for a given consumer and/or producer application but rather more focused on the Kafka cluster holistically and the infrastructure that it runs on. Some specific needs, requirements, and questions from a Platform Operator may include the following:

Platform Ops Use Case	Description
Use Case 1	I would like a single platform to monitor all the Kafka clusters within my organization. I want to be able to quickly switch from one Kafka cluster to another.
Use Case 2	I would like to get quick current snapshot of my cluster: number of producers, number of brokers, number of topics, number of consumers.
Use Case 3	Across the entire cluster, which producers are generating the most data right now?
Use Case 4	Across the entire cluster, which of my consumer groups and consumer instances are falling behind with respect to reading from a topic or partition?
Use Case 5	I would like to see a snapshot view of all the Kafka brokers in my cluster with information including the hosts on which the broker is running, throughput in, messages in, number of partitions, and number of replicas.
Use Case 6	Are any of my brokers running hot? Which broker has the highest throughput in and out rates?
Use Case 7	Which topic partitions are on a given Kafka broker?
Use Case 8	Are there any skewed partitions for a broker? What is the throughput in and out for a given partition on that broker?
Use Case 9	For a given broker, topic, or partition, which producers are sending data to it, and which consumer groups are consuming from it.
Use Case 10	View detailed level metrics of a broker across time to see trends and patterns.
Use Case 11	Whats are host metrics on the host where my broker is running? What are the other services running on my broker host?

Let's walk through how SMM can answer these questions for a Platform Operations user:

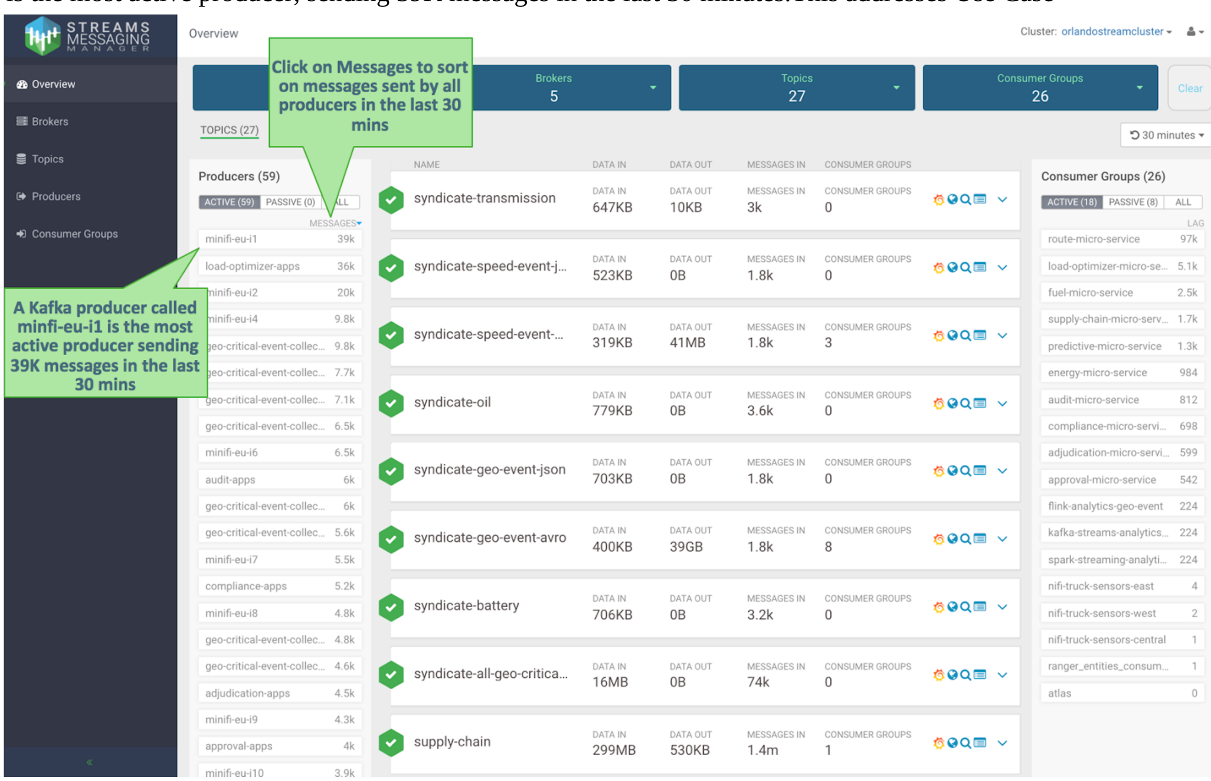
1. Select the Kafka Cluster you want to monitor with SMM. This takes you to the main dashboard view for that cluster selected. This view gives you a powerful snapshot of the workings of the cluster displaying: total number of active and inactive producers and consumers, all topics with summary metrics. This view also provides the ability to filter on four key Kafka entities: producers, brokers, topics, and consumer groups. This addresses

Use Case 1 which is the ability for SMM to manage multiple Kafka clusters in the organization and Use Case



2.

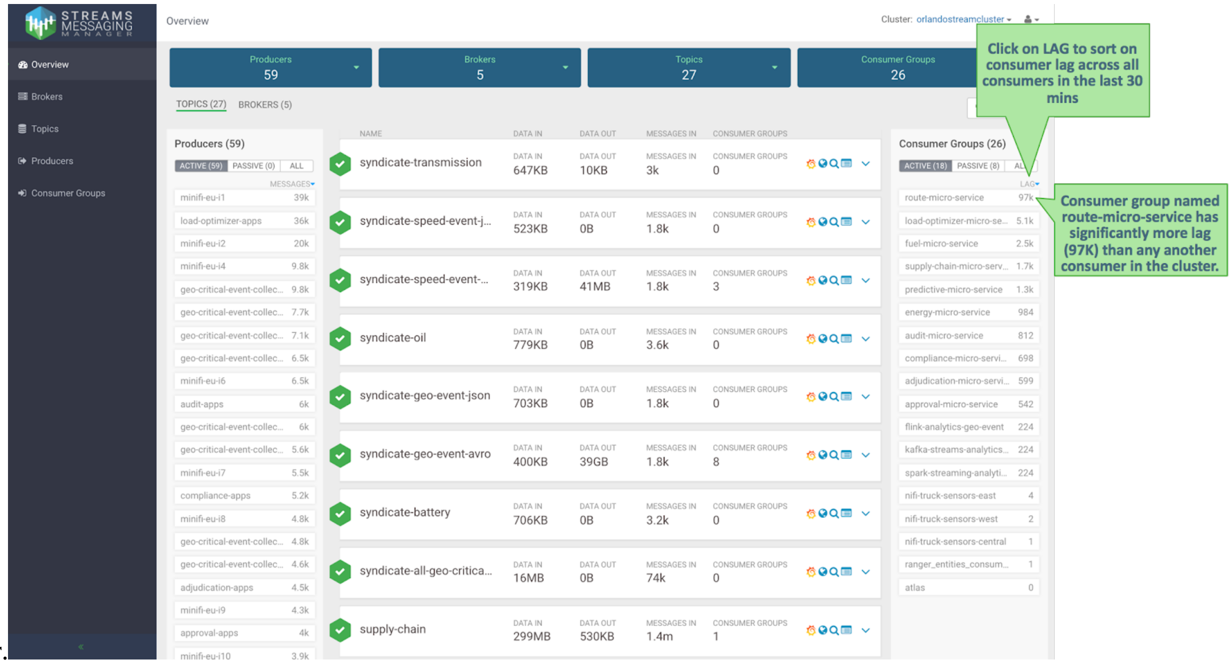
2. From the Producers panel on the left hand side of the screen, select the Messages header to sort on messages sent by all the producers in the system. We see that the Kafka producer called minifi-eu-i1 is the most active producer, sending 39K messages in the last 30 minutes.This addresses Use Case



3.

3. On the Consumer Groups panel on the right hand side, select the column LAG to sort on consumer group lag. This lag is defined as the summation of all consumer instances lag in the consumer group. This addresses

Use Case 4 because we can easily see that we have a consumer group called micro-alert-service that has a lag of 97K over the last 30 minutes which is significantly more than any other consumer group in the



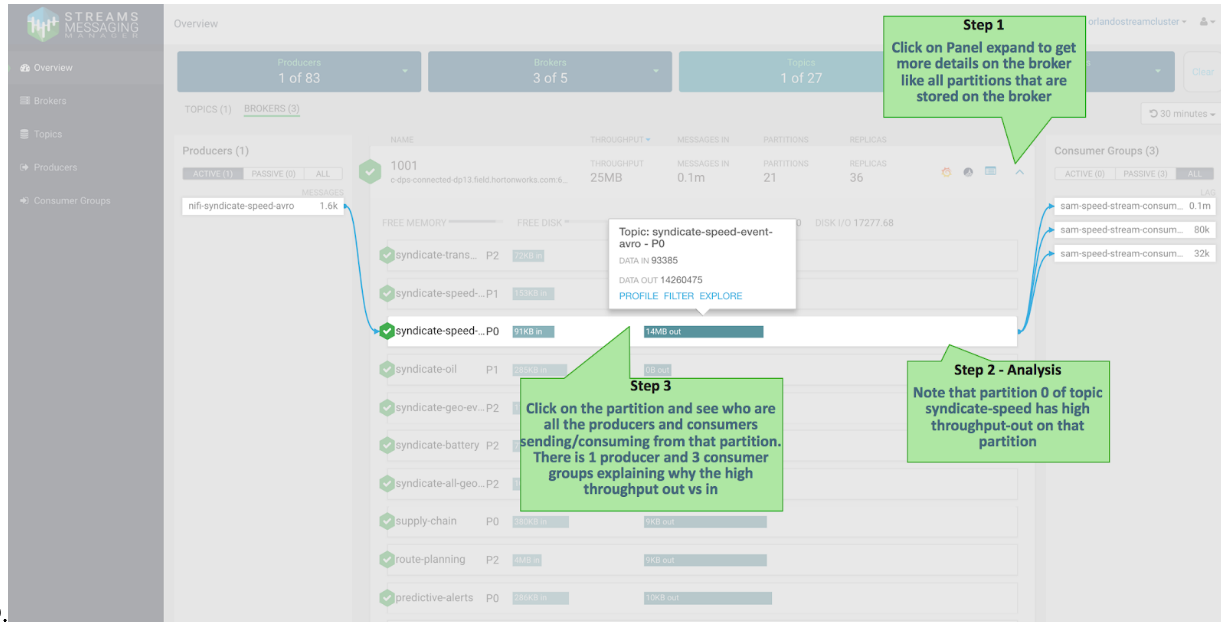
cluster.

4. Click the Brokers tab to see a broker centric view of the dashboard. You can view important metrics for each broker in the cluster including: hosts the broker is running on, throughput in, messages in, number of partitions, and number of replicas. Click the THROUGHPUT column to sort by throughput across all brokers. You can easily see that broker 1001 has the highest rate of data in over the last 30 minutes: 80K messages totaling 17MB. This addresses Use Cases 5 and

6.

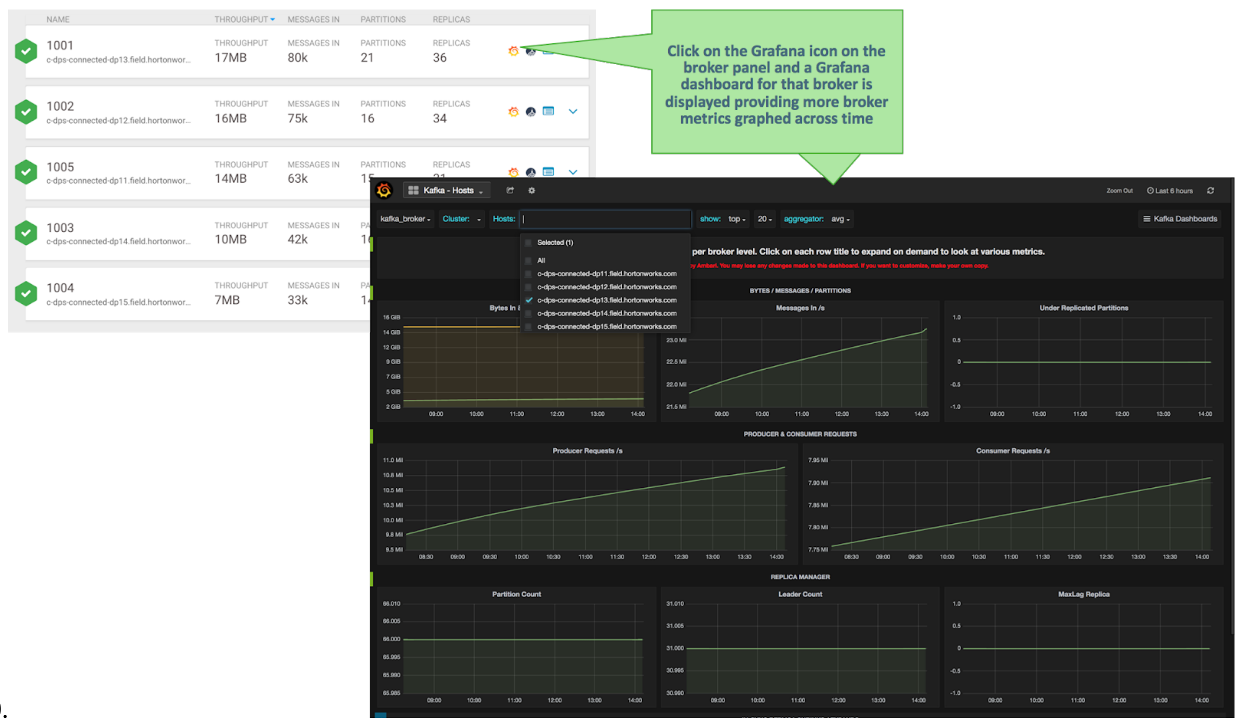
5. Click on the broker panel to expand it and see more details and metrics for that broker. The expanded panel shows all partitions for different topics that are stored on that broker. For each partition, we see the throughput in and out relative to other partitions. We can click on a given partition and see all the

consumers currently sending data to that broker, topic, or partition and all consumer groups consuming from that broker, topic, or partition. We can easily see that partition 0 of topic syndicate-speed on broker 1001 has considerably higher throughput out than any of the other partitions on that broker. By viewing how data flows in and out of that partition, we see that the partition has 1 producer but 3 consumer groups which explains the high throughput out relative to the throughput in. This addresses Use Case 7, 8 and



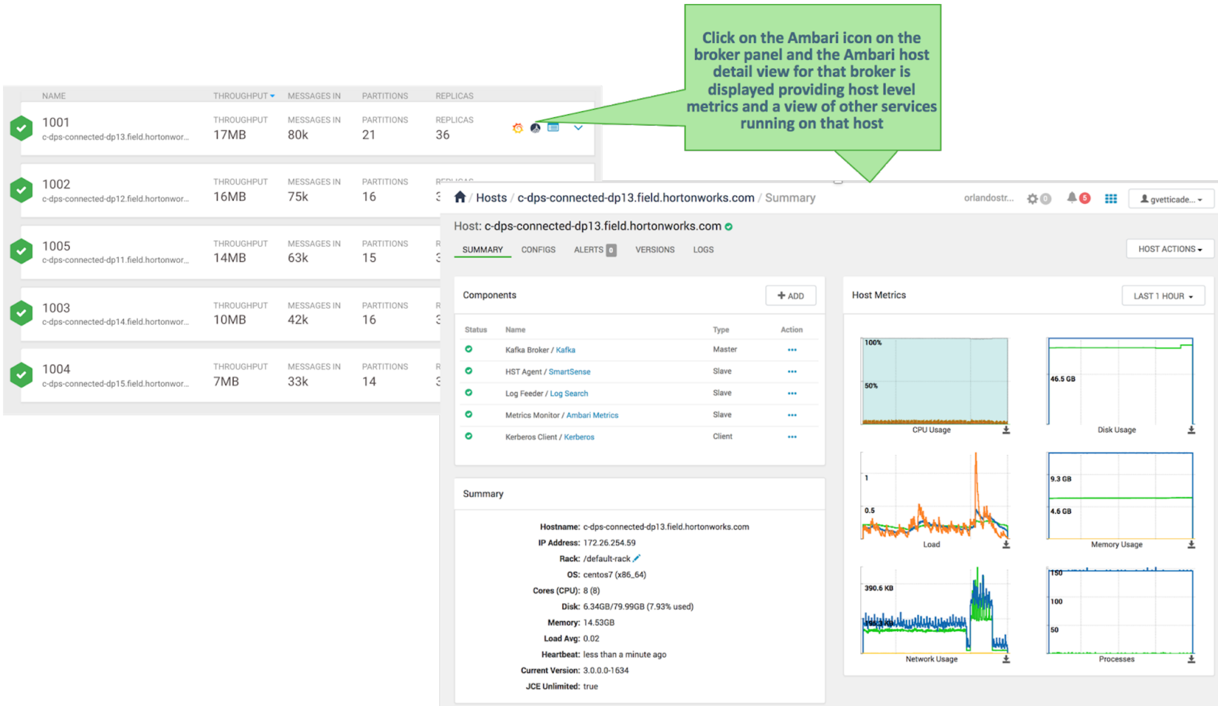
9.

6. Click the Grafana icon on the broker panel to see more detailed metrics within Grafana for that broker. This addresses Use Case



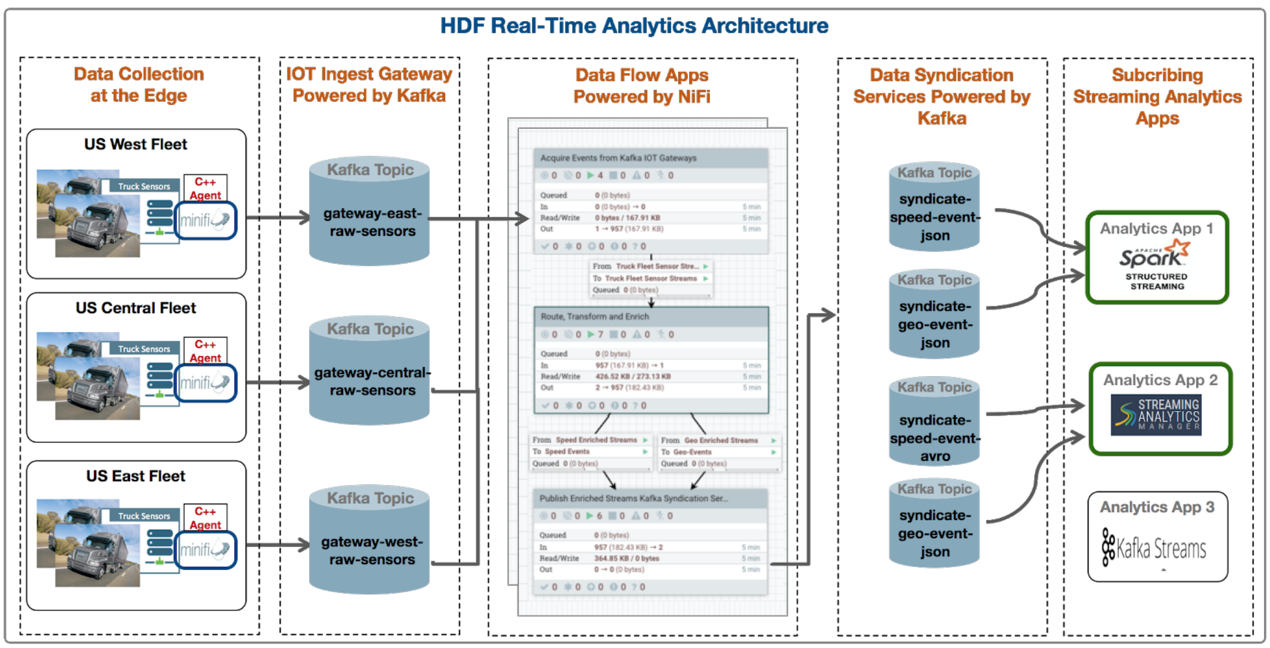
10.

7. Click the Ambari icon on the broker panel and view Broker Host detail metrics. This addresses Use Case 11.



DevOps and Application Developer Persona

Unlike a Platform Operations persona, the DevOps / AppDev persona is most interested in the entities (producers, topics, consumers) specific to their application. So let's assume we are on the DevOps team responsible for monitoring the Trucking Reference Application that is deployed in production, based on this architecture.



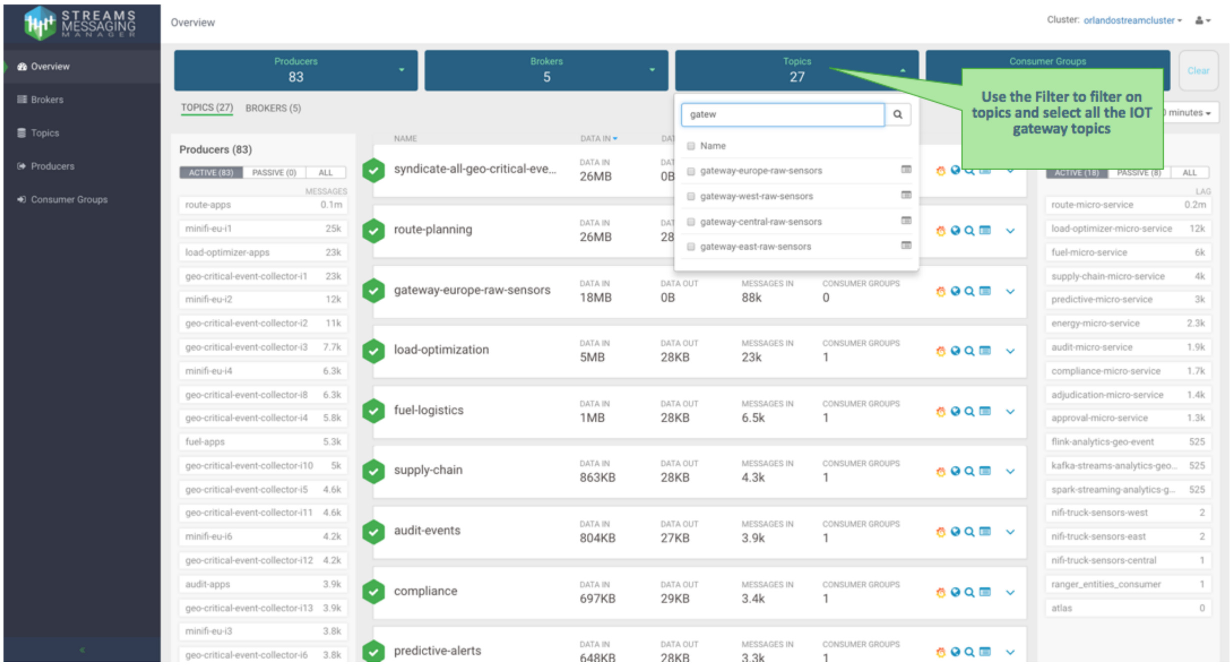
Some specific needs/requirements/questions as a DevOps member for the trucking ref app might be the following.

DevOps/App Dev Use Case	Description
Use Case 1	I want to quickly find all entities (producer, consumers, topics) associated with my application.

Use Case 2	For all the topics associated with my applications, I want to easily see important metrics such as throughput in/out, number of consumer groups, number of messages across a period of time.
Use Case 3	Which of my topics is being sent the most amount of data over a certain period of time? In other words, which regional/geo truck fleet is sending the most amount of data?
Use Case 4	For a given Kafka topic that is part of my application, which are all the connected producers sending data? In other words, which truck fleets are sending data to a gateway topic?
Use Case 5	Are there any topics who have producers but no consumer groups connected to it? In other words, are trucks sending data to a topic but no analytics or processing is being done?
Use Case 6	For a given topic, how many partitions are there? Where are the partitions located? How is data distributed across the partitions? Are there any partition skews?
Use Case 7	Which consumer groups are connected to a given topic and are actively consuming data from it?
Use Case 8	For a given topic, I want to be able to explore data in the topic searching using offsets and/or partition.
Use Case 9	I want to find Metadata and Lineage of the Topic across producers, consumers, and multiple Kafka hops.

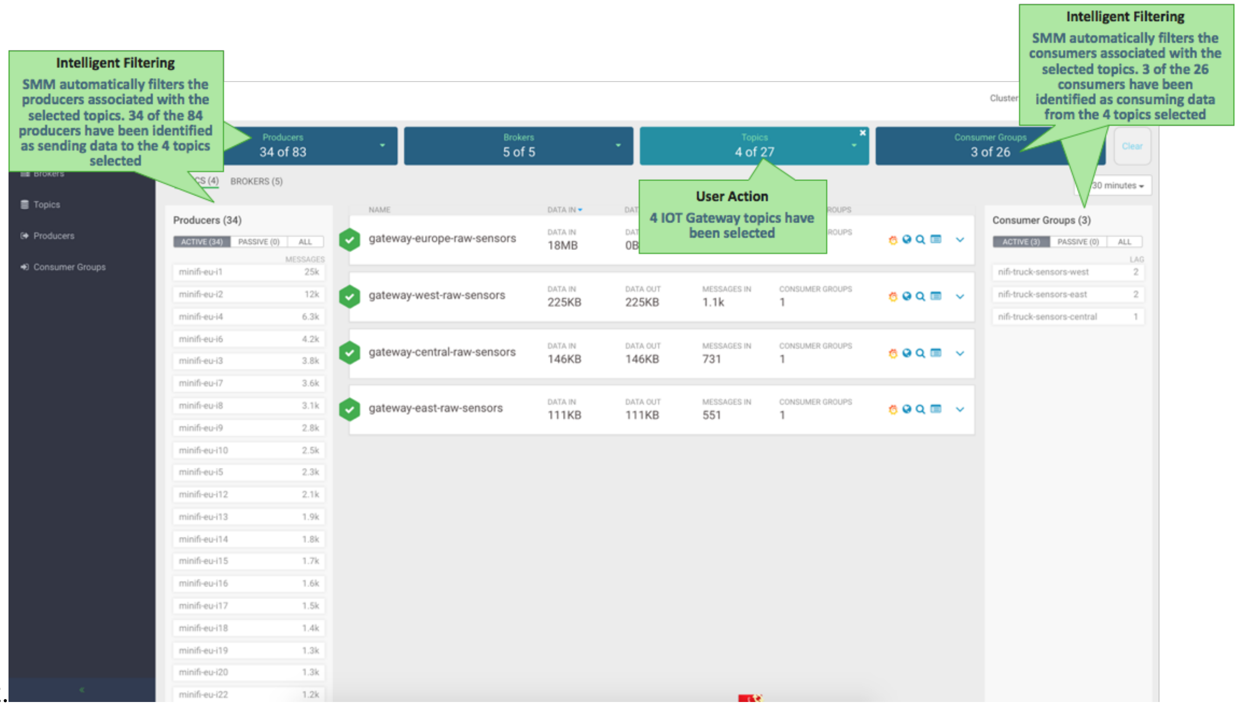
Let's walk through how SMM can answer these questions for a DevOps/App Dev user.

1. Select the Topic filter and select all the IoT gateway topics by searching for all topics that start with gateway.



2. When the filter is applied, SMM provides intelligent filtering by showing only the producers that are sending data to the 4 gateway topics and the consumer groups only consuming data from those topics. So, when the user selects the 4 gateway topics, SMM displays 34 of the 83 producers sending data to those topics and 3 of the 26 consumer groups consuming data from it. Key metrics for the selected topics are shown including

data-in and out, number of messages, number of consumer groups, etc. This addresses Use Cases 1 and

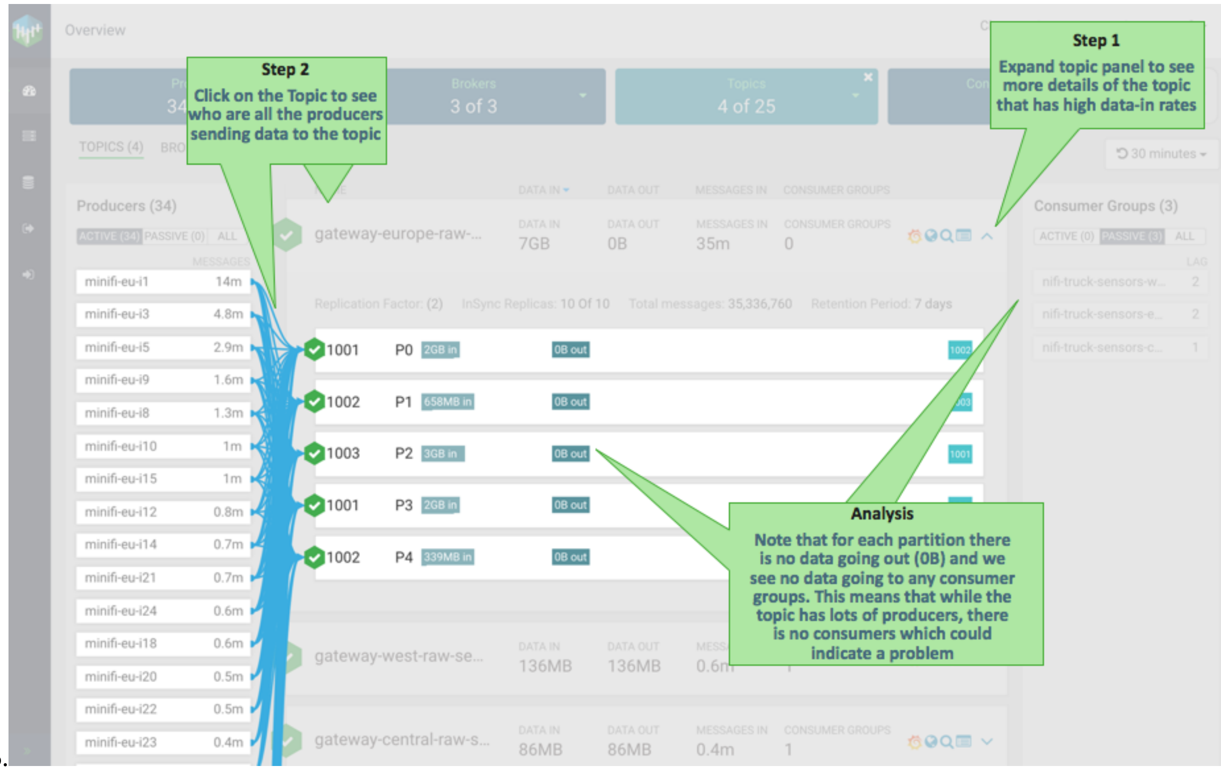


2.

3. Click DATA IN to sort on data throughput-in across all topics. We see that gateway-europe-raw-sensor has significantly more data coming in than any other topic: 18 MB totaling 88K in the last 30 minutes. This addresses Use Case 3.

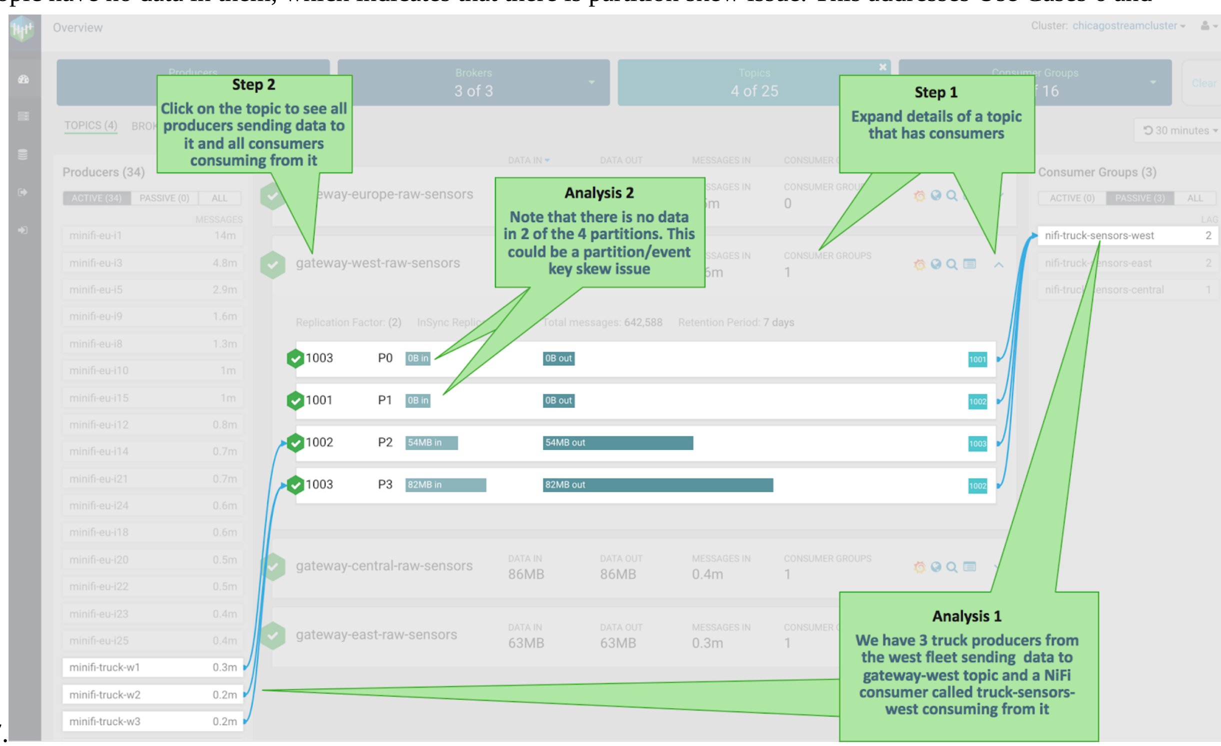
4. Expand the topic panel for gateway-europe-raw-sensor to get more details for the topic, including like partition layout. Click the topic to see which producers are sending data to each partition of that topic. We see that all the producers are trucks from the EU fleet. Also note that all five partitions for that topic have 0B going out and we see no data flowing from the topic to any consumer group. This could be worth investigating to

identify why a topic with the most amount of producers has no consumers. This addresses Use Cases 4 and



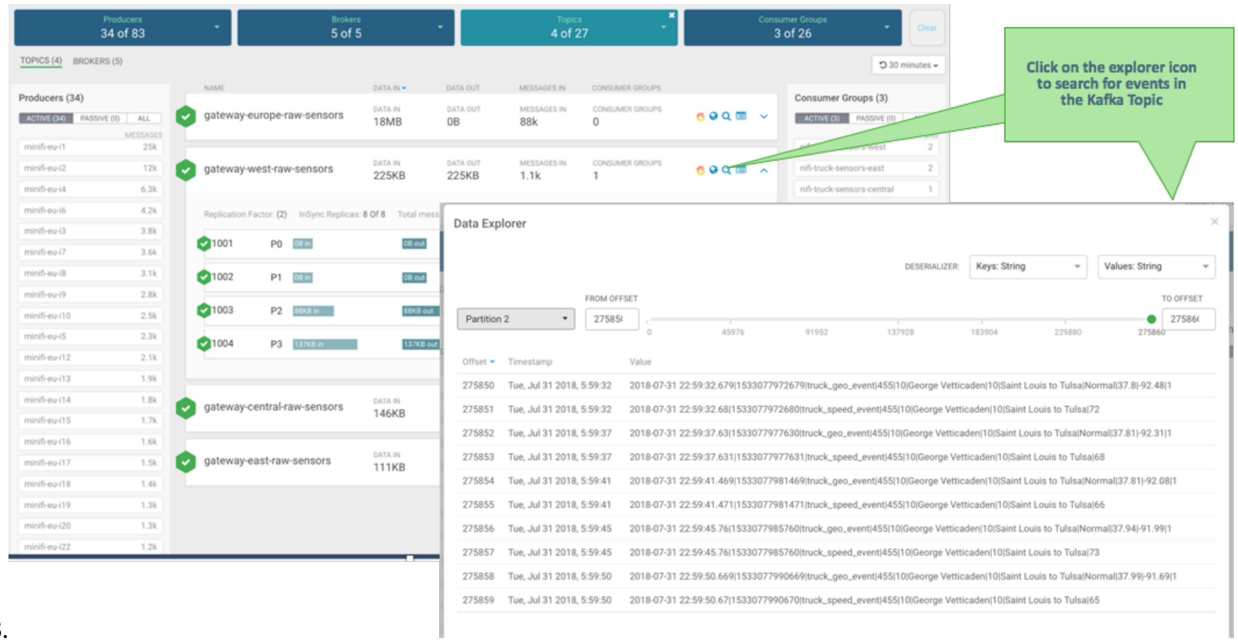
5.

5. Click another topic called gateway-west-raw-sensors that has consumers. This topic has three producers sending data to it and a NiFi consumer consuming from it. We also see that two of the four partitions for this topic have no data in them, which indicates that there is partition skew issue. This addresses Use Cases 6 and



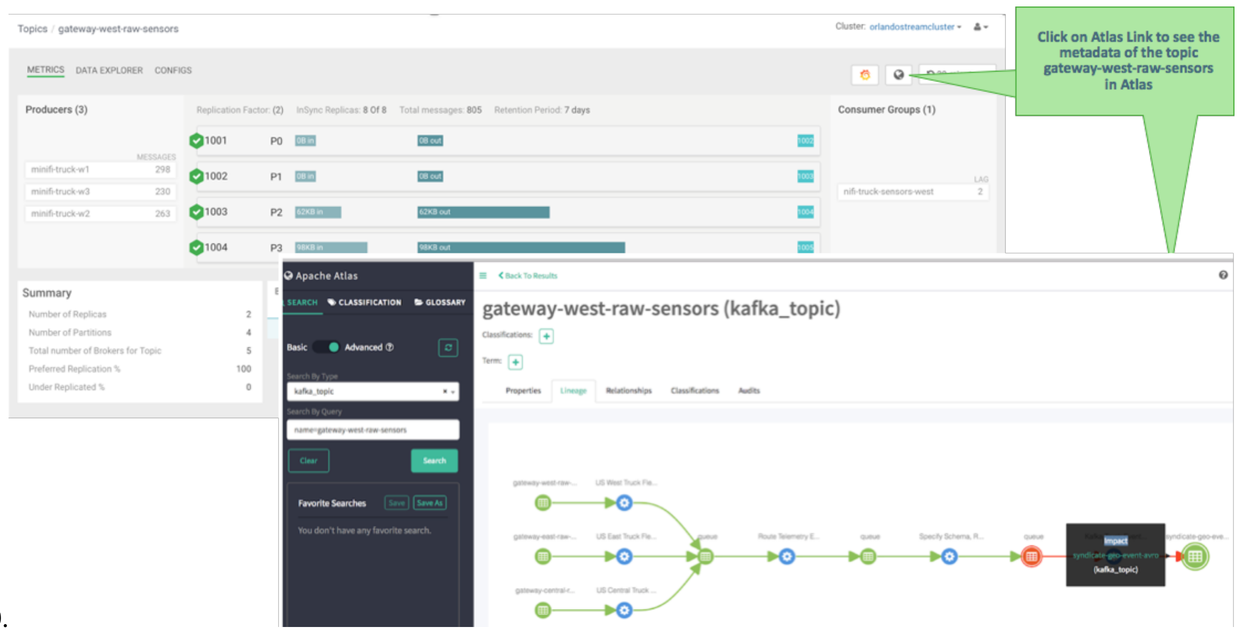
7.

6. Click the explorer/magnifying glass icon to search for events in the selected Kafka topic. This addresses Use Case



8.

7. Click the Atlas icon to see the metadata and lineage of the Kafka topic in Atlas. This addresses Use Case



9.