Hortonworks DataFlow

Hortonworks Streaming Analytics Manager User Gu

(June 9, 2017)

docs.hortonworks.com

Hortonworks DataFlow: Hortonworks Streaming Analytics Manager User Guide

Table of Contents

List of Tables

1. Streaming Analytics Manager Environment Setup and Managing Stream Apps

The information in this chapter focuses on the following operational tasks, suited for the operations persona. When you access Streaming Analytics Manager (SAM) for the first time, you must perform two operations tasks to get started

- Creating service pools
- Creating environments

Subsequent subsections walk through each of these steps.

1.1. Managing Service Pools

A **Service** is an entity that an application developer works with to build stream apps. Examples of services include a Storm cluster to which you want to deploy the stream application, a Kafka cluster that stream application uses to create a streams, or an HBase cluster to which the stream application writes.

A **Service Pool** is a set of services associated with an Ambari managed cluster. To manage service pools, hover over the **Configuration** tab and select the Service Pool menu item.

The Service Pool dashboard lists all existing service pools, and allows you to create new service pools.

1.1.1. Adding a New Service Pool

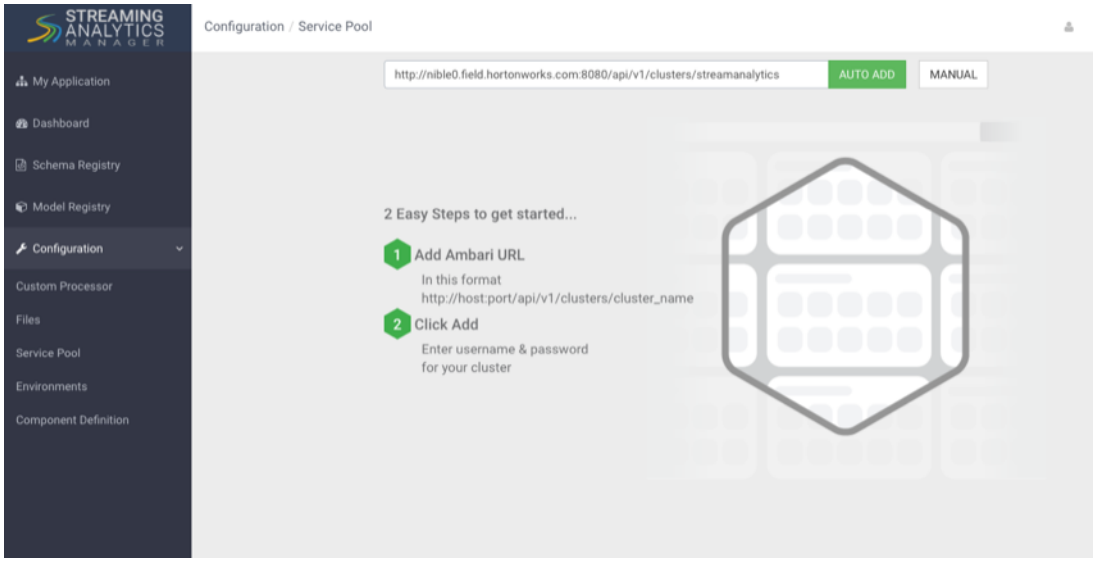
Prerequisites

You have deployed an Ambari-managed HDF or HDP cluster.

Steps

1. From **Configuration / Service Pool**, enter the rest endpoint URL for your Ambari managed cluster.

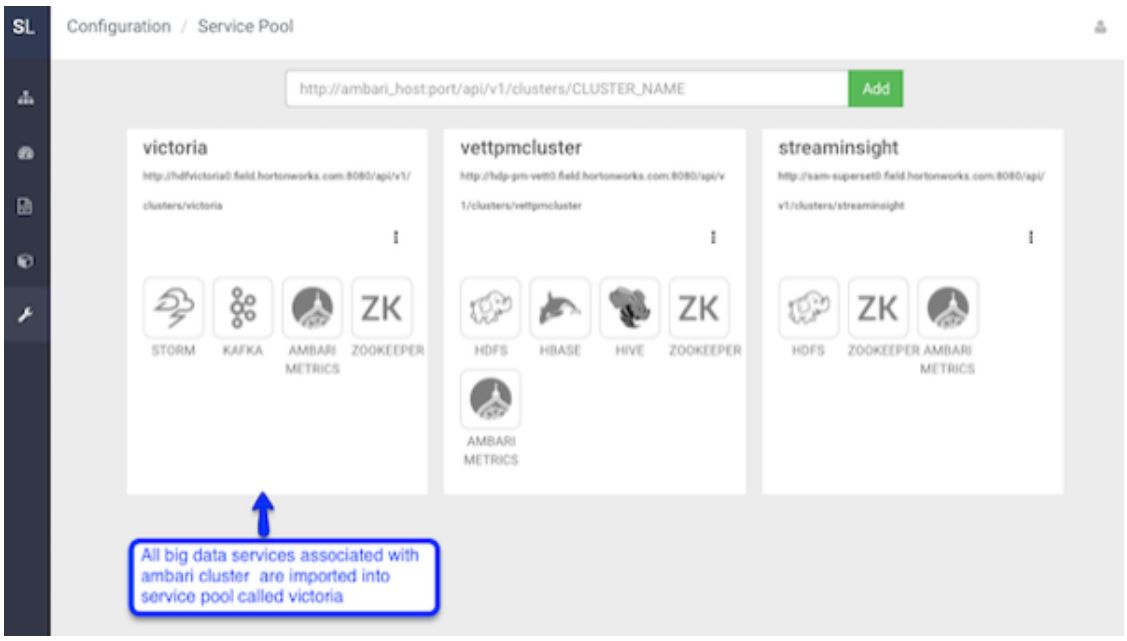
The syntax of this URL has the following form: `http://[AMBARI_HOST]:8080/api/v1/clusters/[AMBARI_CLUSTER_NAME]`.



2. Click **Auto Add**.
3. You are prompted for Ambari credentials. Enter a valid username and password.

Result

SAM retrieves all of the services and creates a new pool. The name of the service pool is the name of the Ambari cluster.



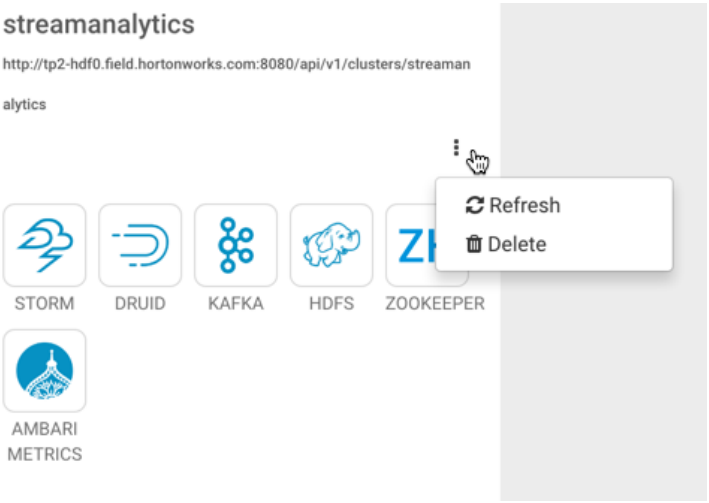
1.1.2. Updating Service Pools

About This Task

When a service pool is created, all of the configuration to manage and connect to the big data services in the pool are imported from Ambari into SAM. If a configuration associated with a service is changed in Ambari, you must manually update the service pool as well.

Steps

1. From **Configuration / Service Pool**, click the **Options** dialog inside the service pool you want to refresh.

2. 

3. Click **Refresh**.

4. Provide your Ambari credentials and click **Ok**.

1.2. Managing Environments

An **environment** is a named entity that represents a set of services chosen from different service pools. A stream application is assigned to an environment. The application can only use the services associated with that environment.

To manage environments, hover over **Configuration** and select **Environments**.

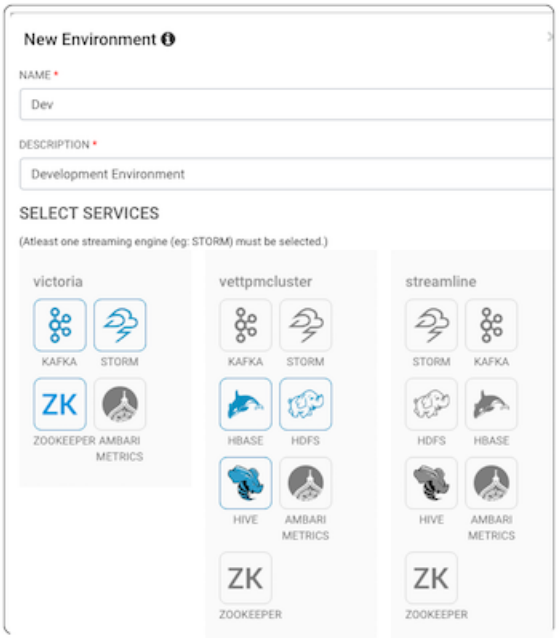
The Environments dashboard lists all existing environments, and allows you to create a new Environment.

1.2.1. Create New Environment

To add a new environment:

1. From **Configuration / Environments**, click the + icon.

2. Name the environment, choose the services that you want in the environment, and click **Ok**. Selected services are highlighted in blue.



Next Steps

After an Environment is created, an application developer can create new stream applications, associate it with the environment, and use the big data services with the application.

More Information

Building an Application

1.2.2. Editing Environments

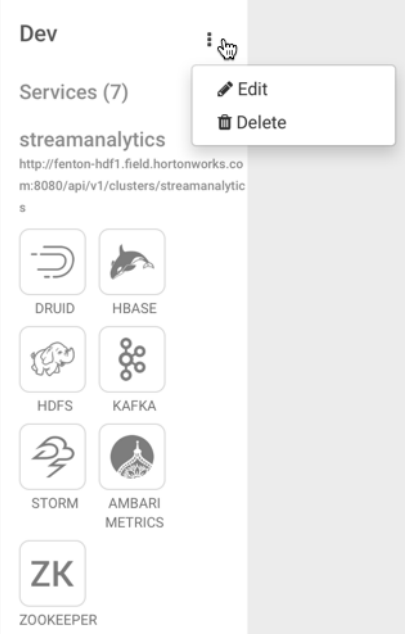
About This Task

You can edit environments by clicking the **Options** icon in the environment box you want to edit.

When an environment is associated with an application, it cannot be deleted or updated.

Steps

1. From **Configuration / Environments**, click the **Options** icon for the envrionment you want to edit.



2. The **Edit Environment** dialog displays. Add additional services or update the name and description of the environment and click **Ok**.

1.3. Deleting Environments

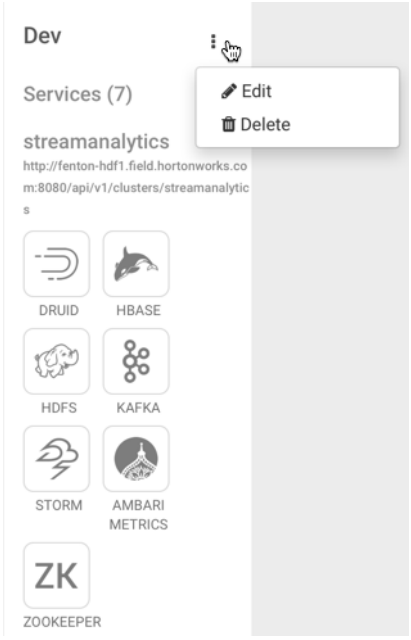
About This Task

You can delete environments by clicking the **Options** icon in the environment box you want to delete.

When an environment is associated with an application, it cannot be deleted or updated.

Steps

1. From **Configuration / Environments**, click the **Options** icon for the envrionment you want to delete.



2. Click **Ok** to confirm you want to delete your environment.

2. Building an Application

Prerequisites

- You have integrated SAM
- You have set up appropriate environments and service pools

Use the following tools to build your stream applications.

- Launch the Stream Builder UI
- Add a New Stream Application
- Add a Source
- Connect Components
- Join Multiple Streams
- Filtering Events in a Stream
- Using Aggregate Functions over Windows

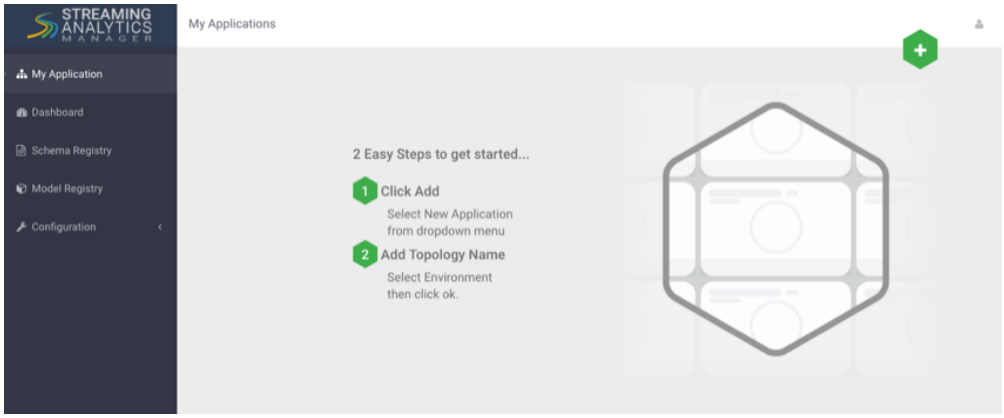
2.1. Launch the Stream Builder UI

Steps

1. In Ambari, select **Streaming Analytics Manager** from the left-hand **Services** pane.
2. Under **Quick Links**, select **SAM UI**.

Result

The SAM Stream Builder UI displays. You can return at any time by clicking **My Applications** from the left-hand menu.



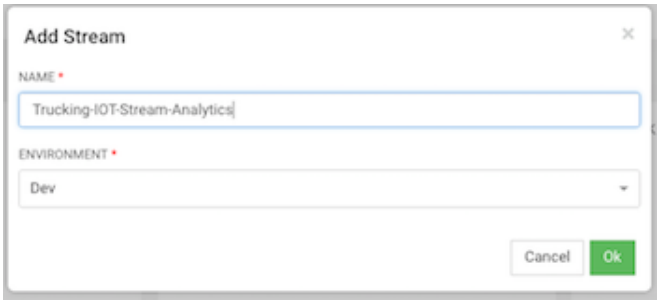
2.2. Add a New Stream Application

Steps

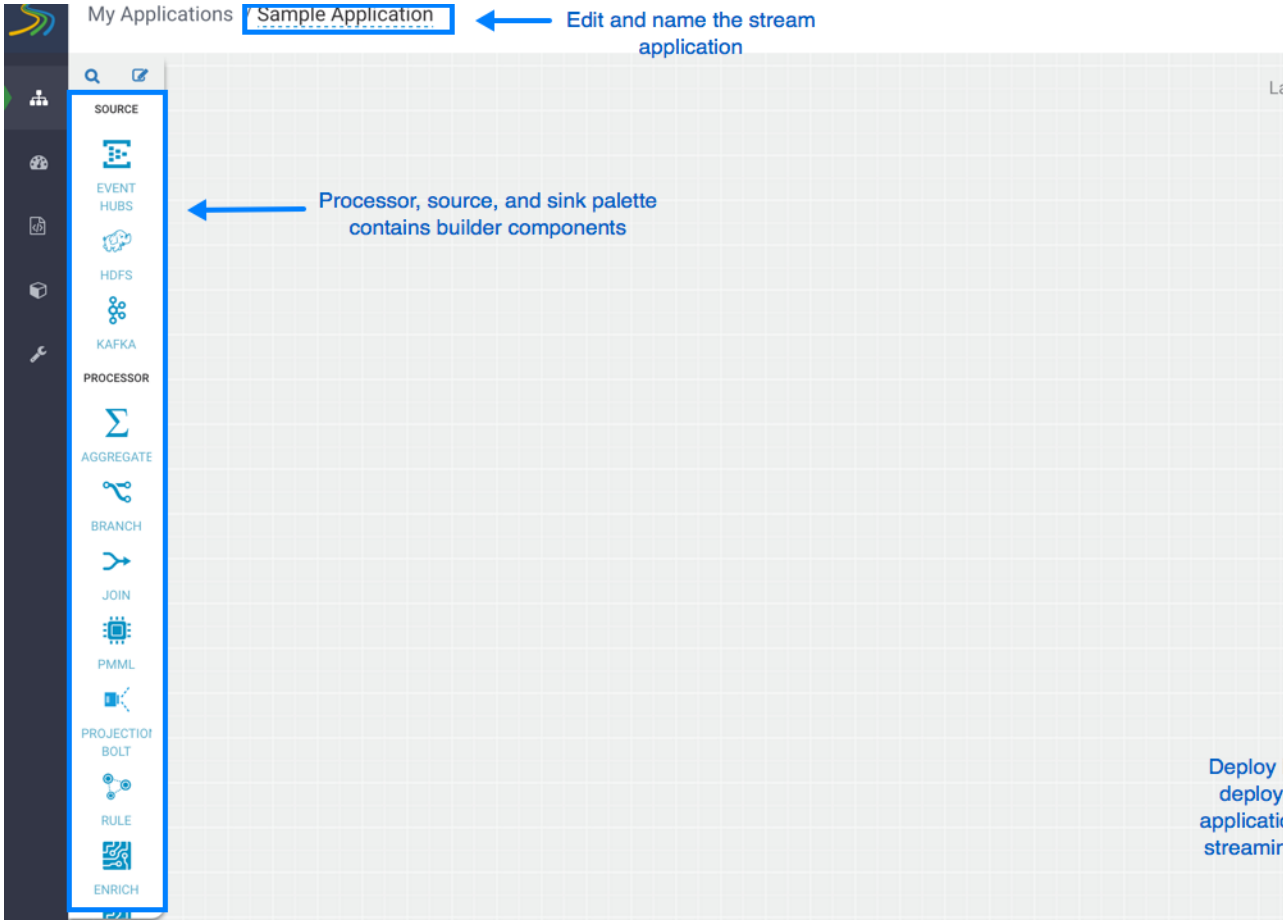
1. Specify the name of the stream application and the environment you want to use.

Note

The name of the stream app should not have any spaces.



2. SAM displays the Stream Builder canvas. Builder components on the canvas palette are the building blocks you use to build stream apps. Refer to the *HDF Overview* for information about each component building block.



More Information

Component Building Blocks

2.3. Add a Source

About This Task

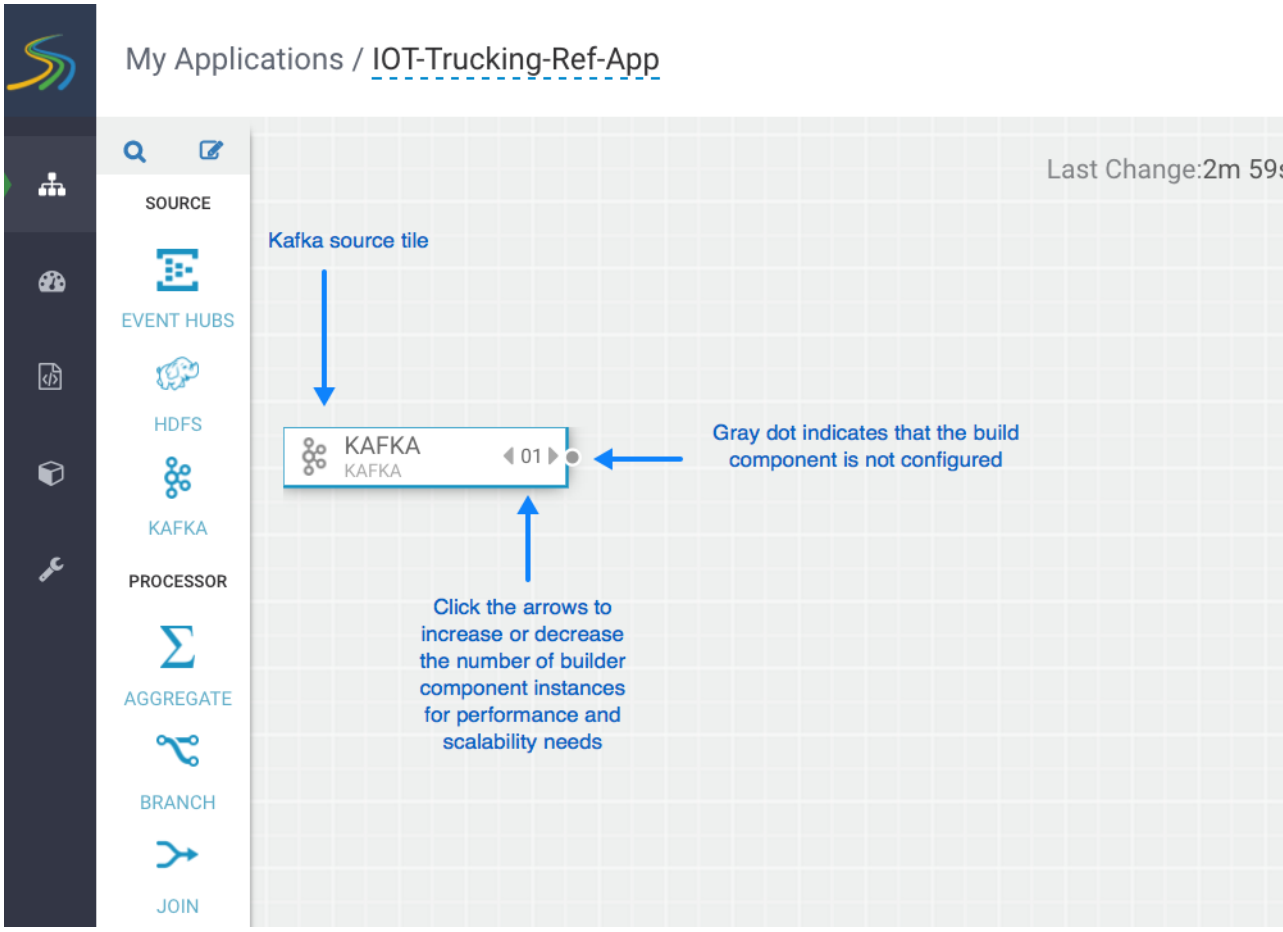
As described in the *HDF Overview*, Stream Builder offers four types of builder components: sources, processors, sinks, and custom components. Start building your application by adding a source.

Prerequisites

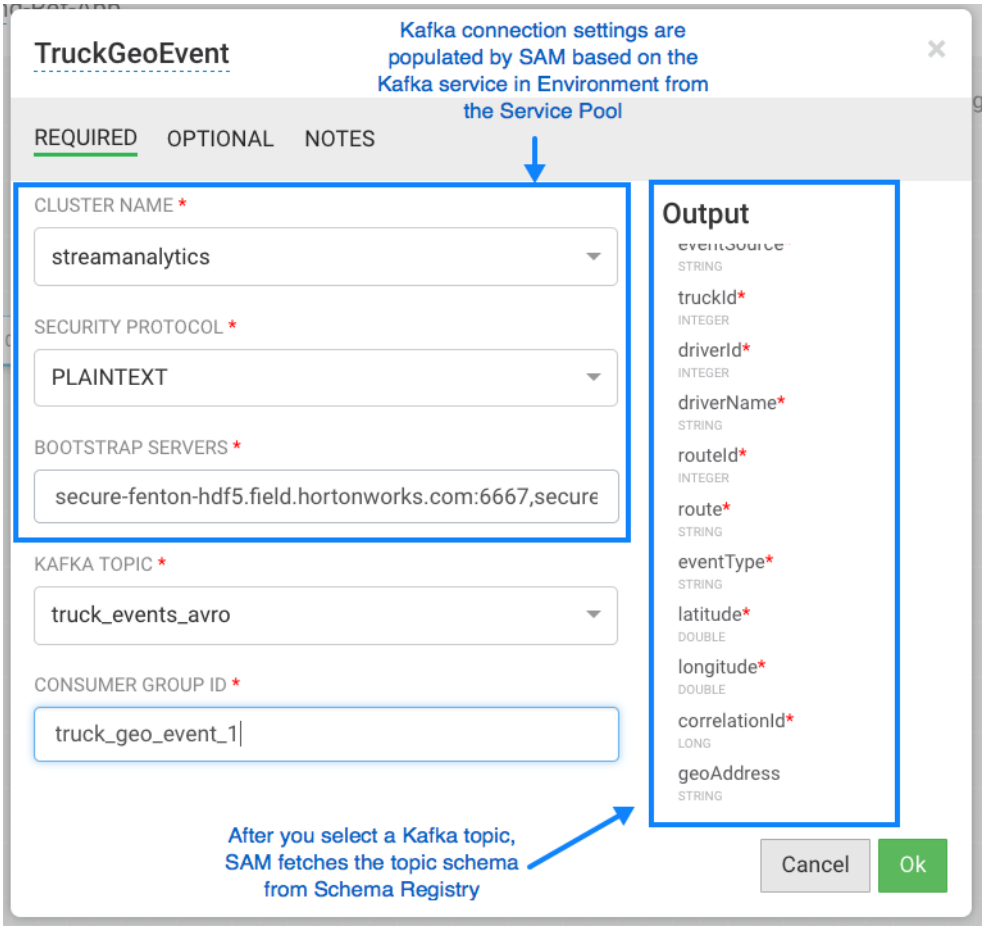
You have configured Schema Registry and integrated with SAM.

Steps

1. Drag a source builder component, Kafka for example, onto the canvas. This creates a Kafka tile component:



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



3. Add the additional components you want to use to develop your stream app.

Result

When you have added and correctly configured your stream app components, the component tile displays a green dot on the left. You cannot connect a source to different proessors or sinks until it is correctly configured.

More Information

Component Building Blocks

Integrating Schema Registry with SAM

2.4. Connect Components

About This Task

Once you have added and configured your source, add additional processors and sinks to the canvas. 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.

Prerequisite

You have added and configured at least one source.

Steps

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

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

2.5. Join Multiple Streams

About This Task

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

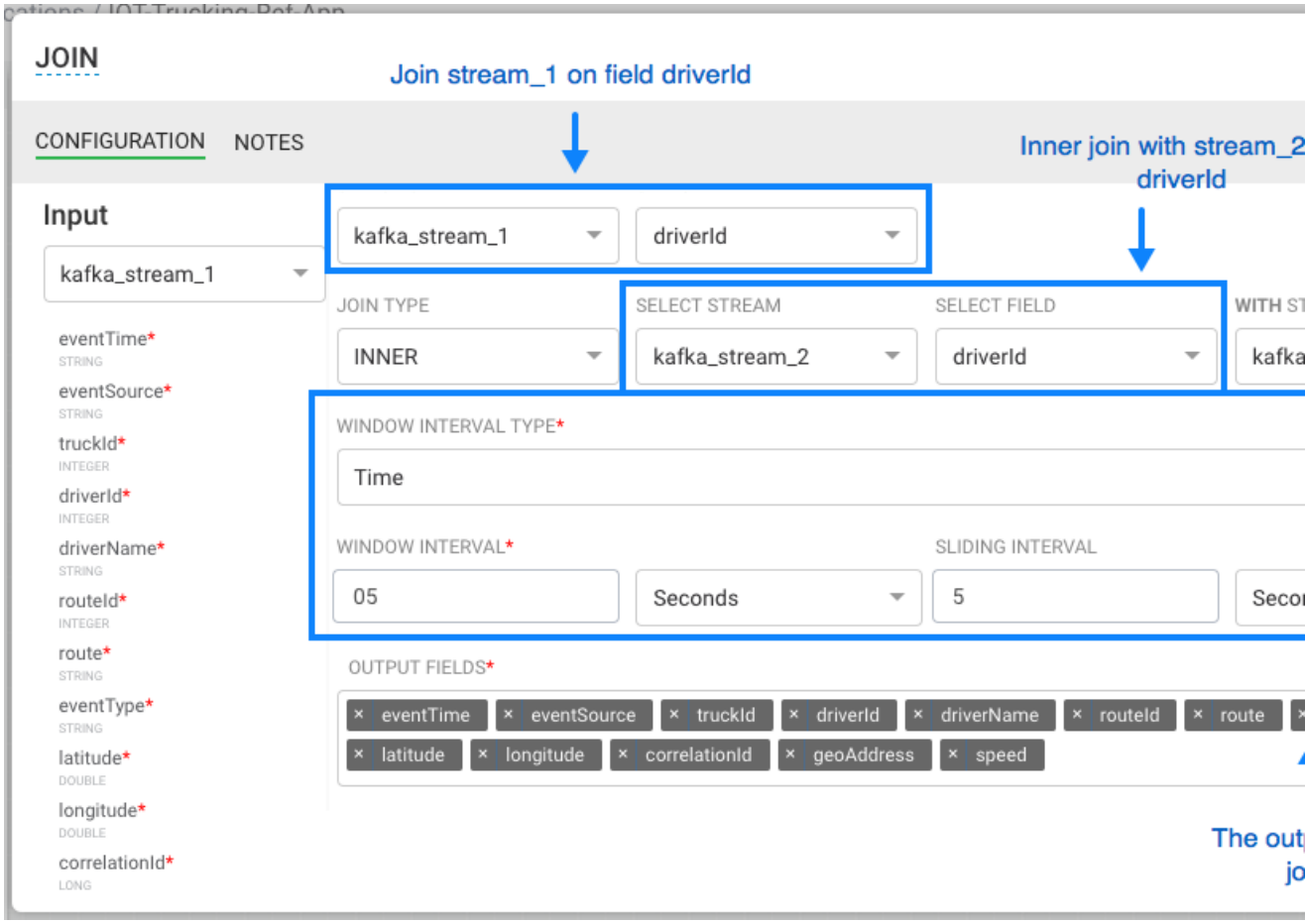
Steps

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

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

3. Configure the Join processors according to your streaming application requirements.

Example



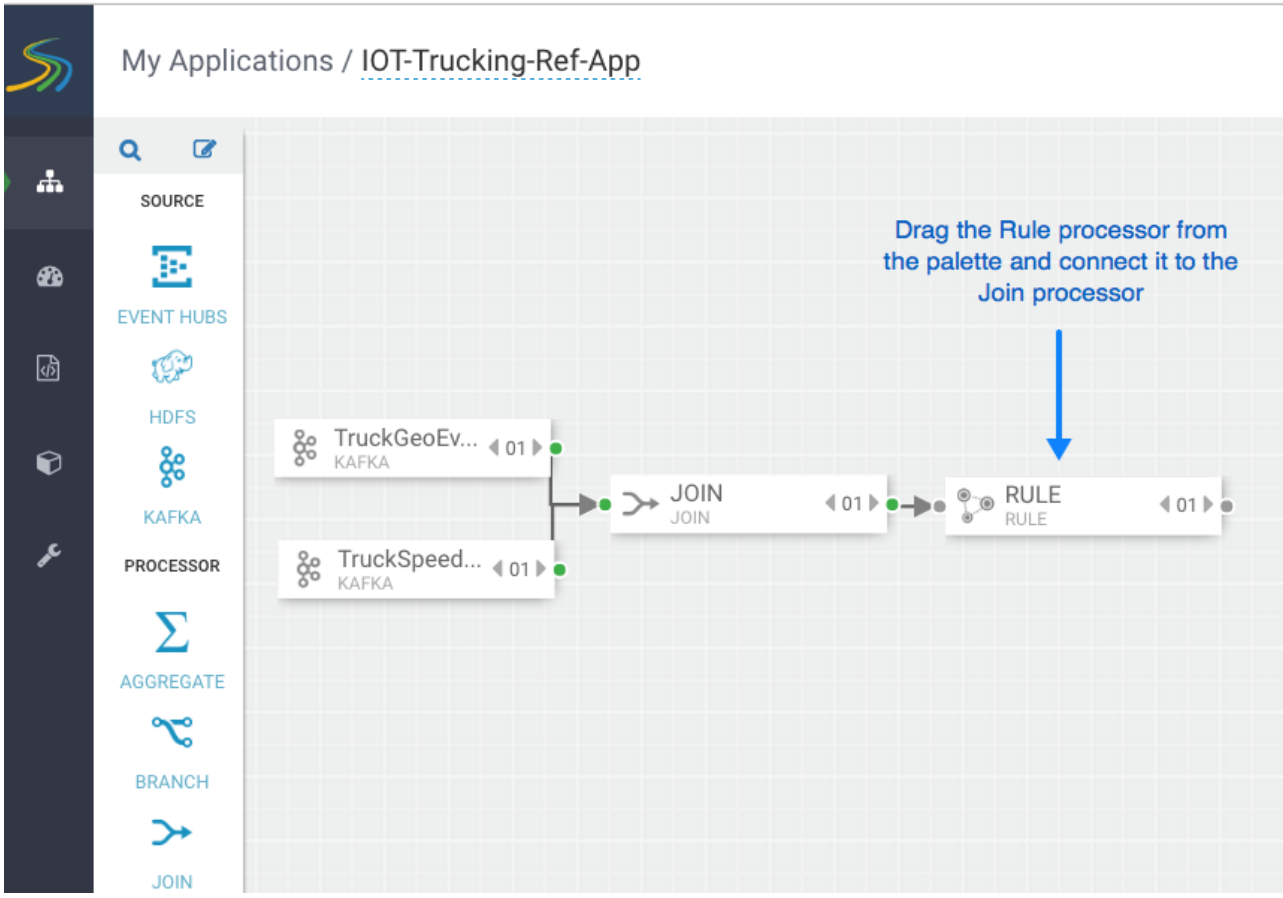
2.6. Filter Events in a Stream

About This Task

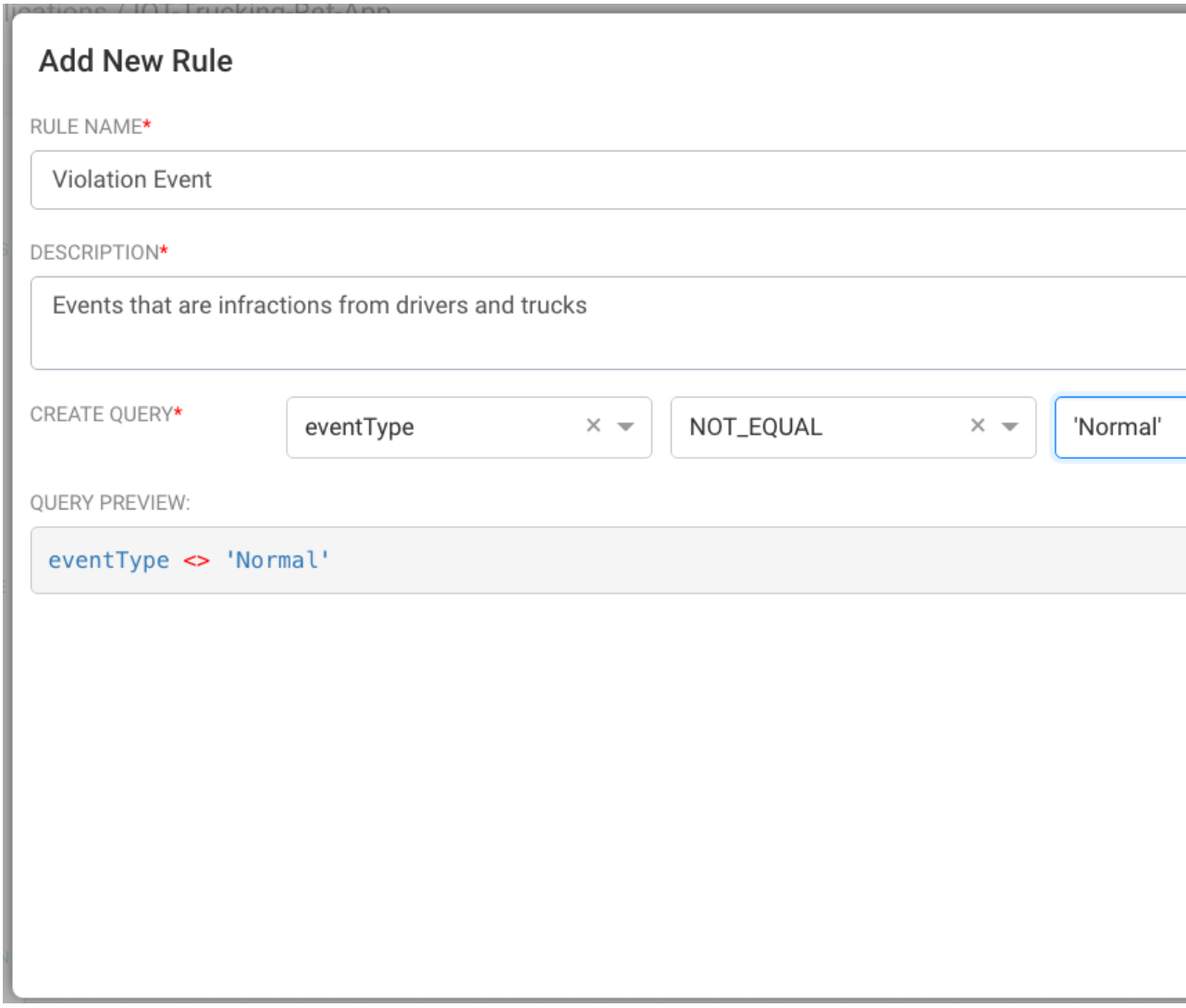
You can use SAM to filter events in the stream. You accomplish this by using Rule processor, which translates rules into SQL queries that operate on the stream of data.

Steps

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

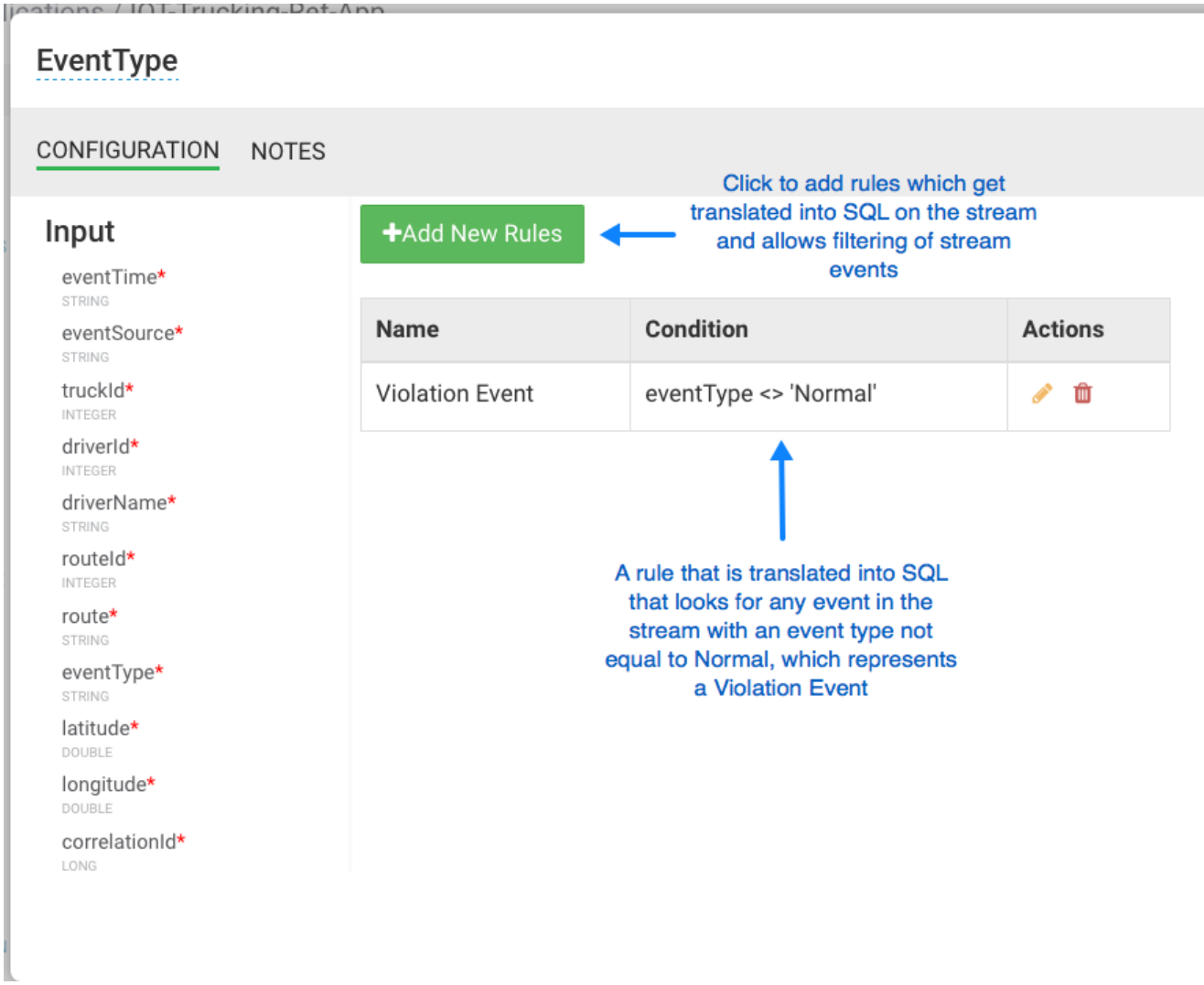


2. Double click the Rule processor, click the + **Add New Rules** button, and create a new rule:



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

Example



2.7. Use Aggregate Functions over Windows

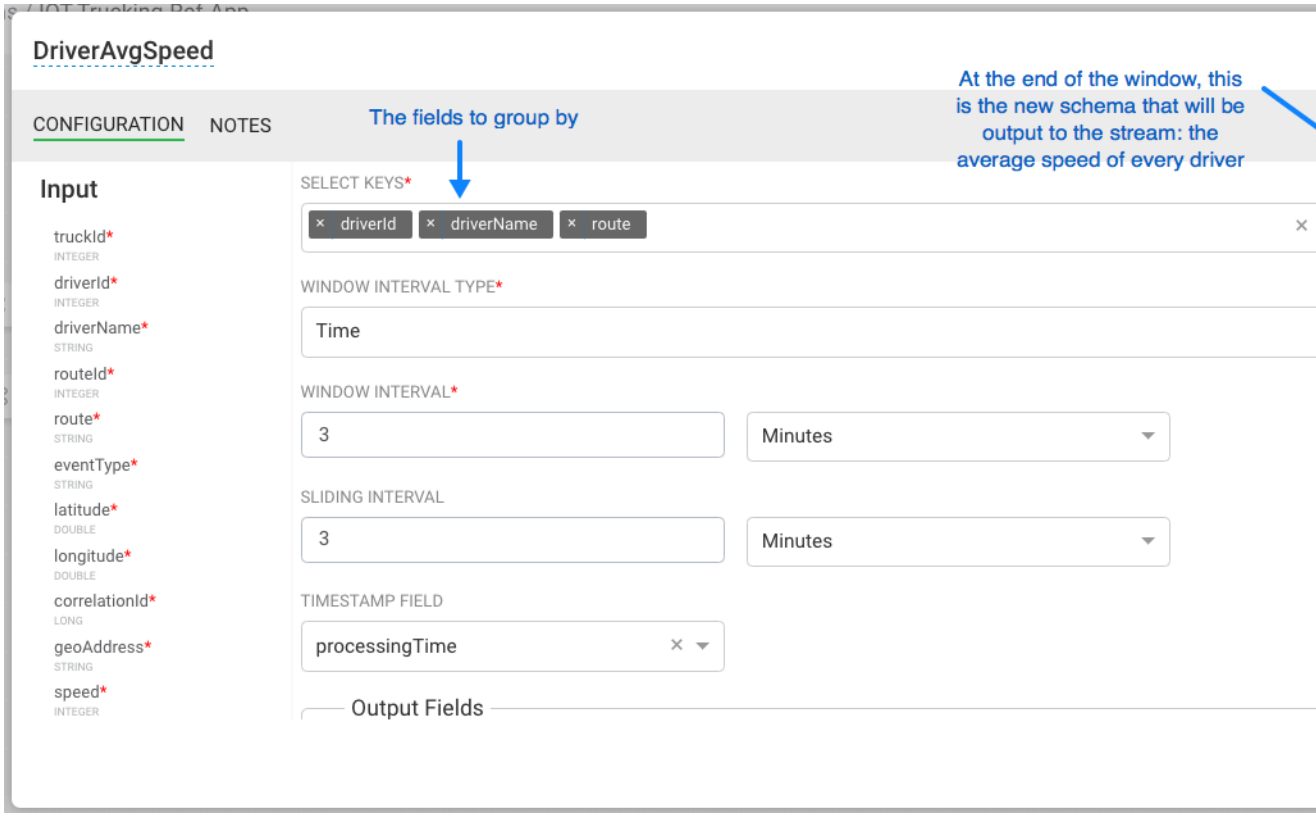
About This Task

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

Steps

1. Drag the Aggregate processor to the canvas and connect it to the stream application you are building.
2. Double click the Aggregate tile to configure it according the your stream application requirements.

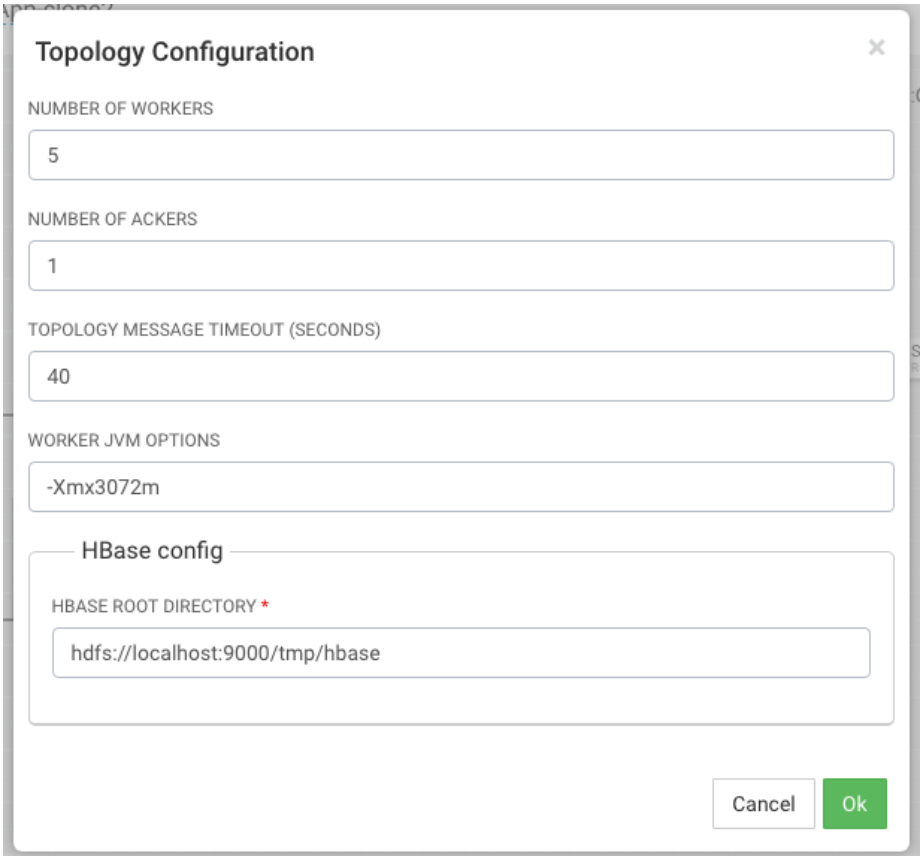
Example



2.8. Deploying a Stream App

2.8.1. Configure Deployment Settings

Before deploying the application, it is important to 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. Click the gear icon on the top right corner of the canvas, and increase the number of workers (e.g: 5) and increase the JVM heap memory (-Xmx3072m).



2.8.2. Deploy the App

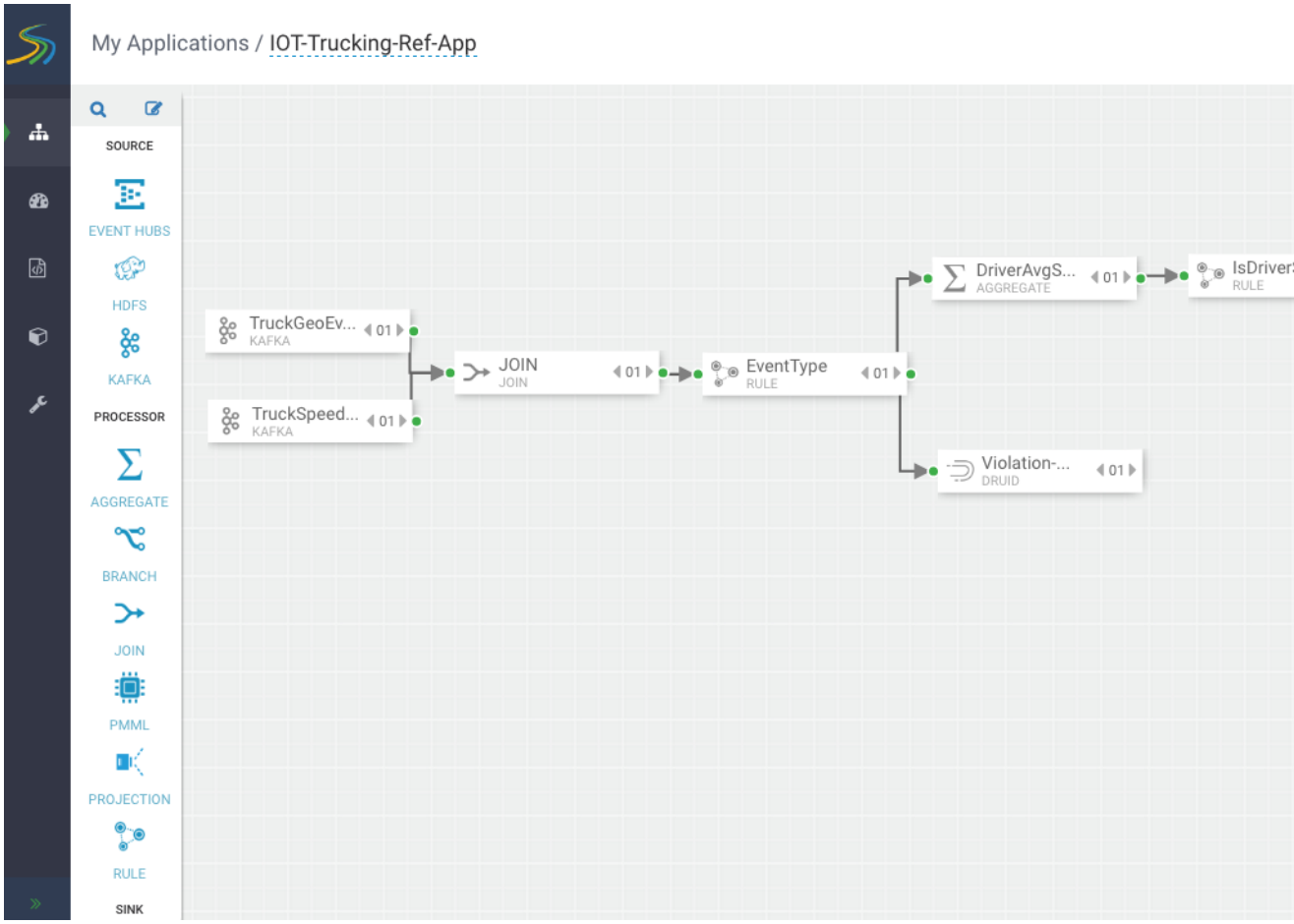
After the app's deployment settings has been configured, click the Deploy button on the lower right of the canvas. During the deployment process, Streaming Analytics Manager completes the following tasks:

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

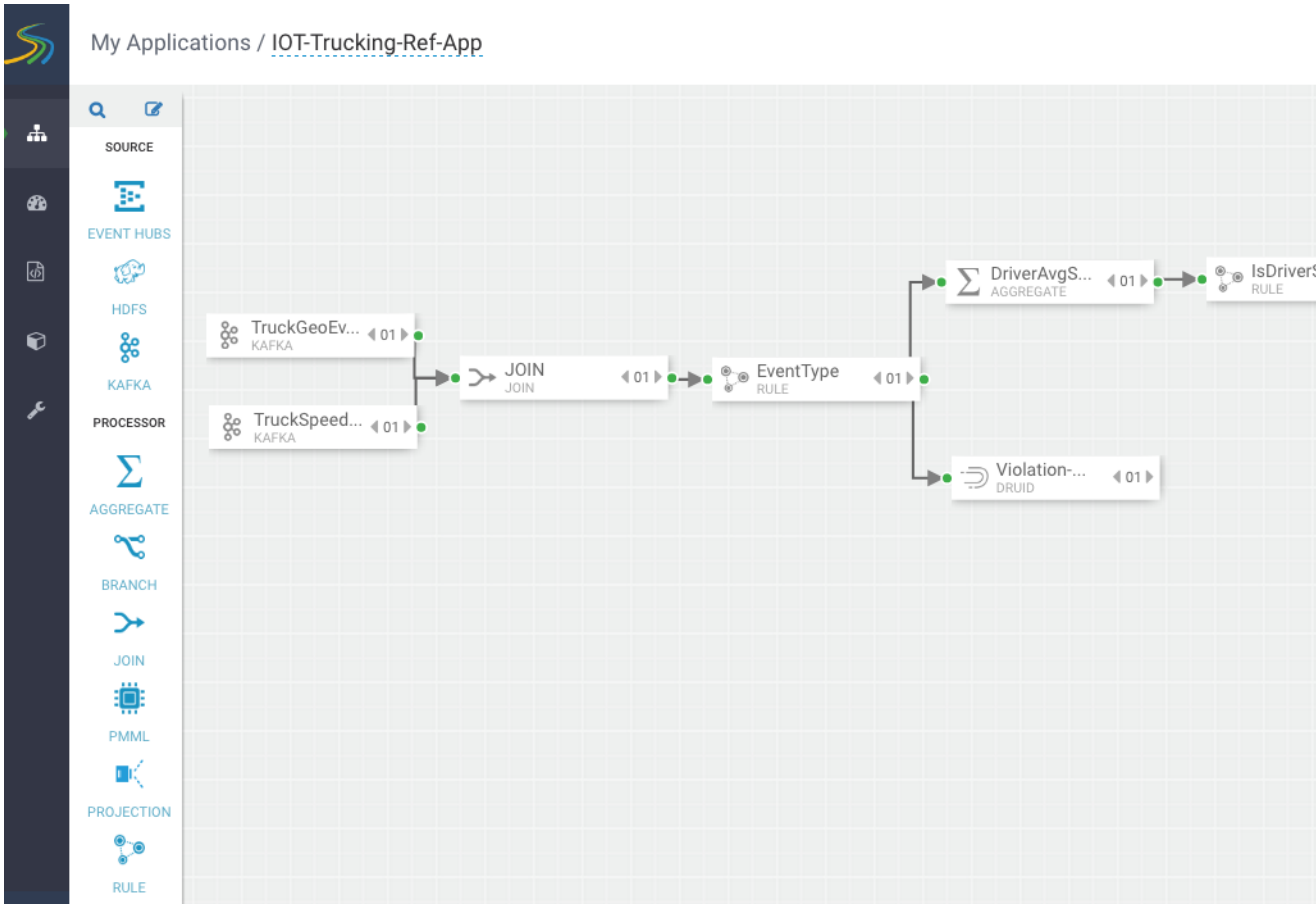
2. Create a deployable jar of the streaming app.

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

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



After the application has been deployed successfully, Streaming Analytics Manager notifies you and updates the status to Active, as shown in the following diagram.


My Applications / IOT-Trucking-Ref-App
SOURCE
EVENT HUBS
HDFS
KAFKA
PROCESSOR
AGGREGATE
BRANCH
JOIN
PMML
PROJECTION
RULE
TruckGeoEv... KAFKA 01
TruckSpeed... KAFKA 01
JOIN JOIN 01
EventType RULE 01
DriverAvgS... AGGREGATE 01
Violation-... DRUID 01

3. Creating Visualizations Using Superset

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

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

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

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

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

Note

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

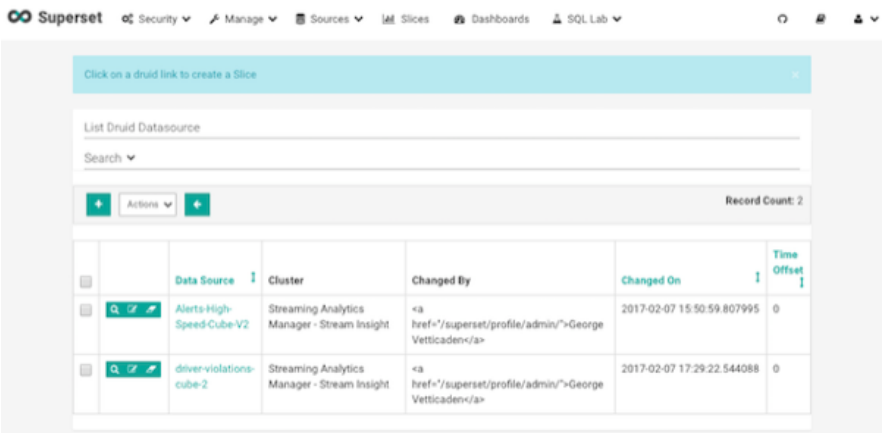
3.1. Creating Insight Slices

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

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

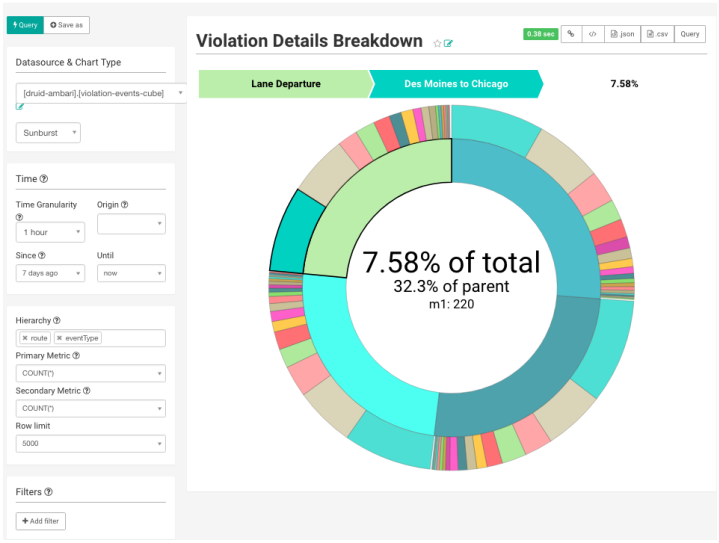
2. Click + to create a new Slice.

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

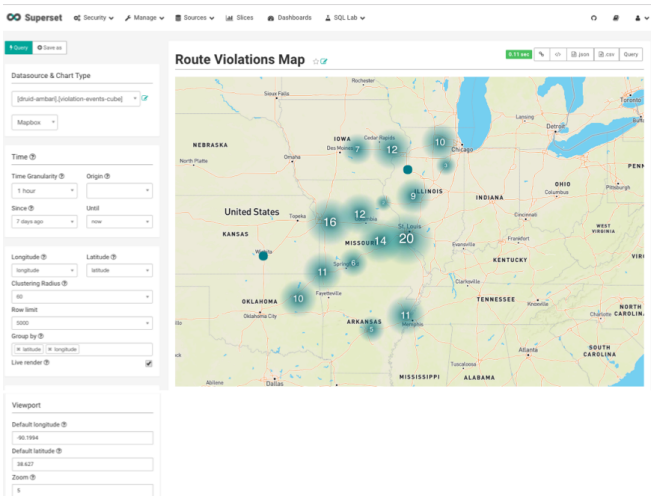


4. Select a Chart Type from the menu.

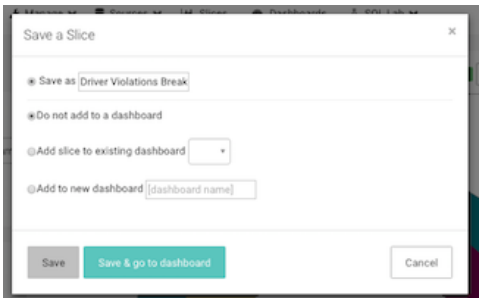
This example creates a "Sunburst" visualization where we are rolling up multiple dimensions like route, eventType and driver info. Configure the chart and click **Execute Query**.



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



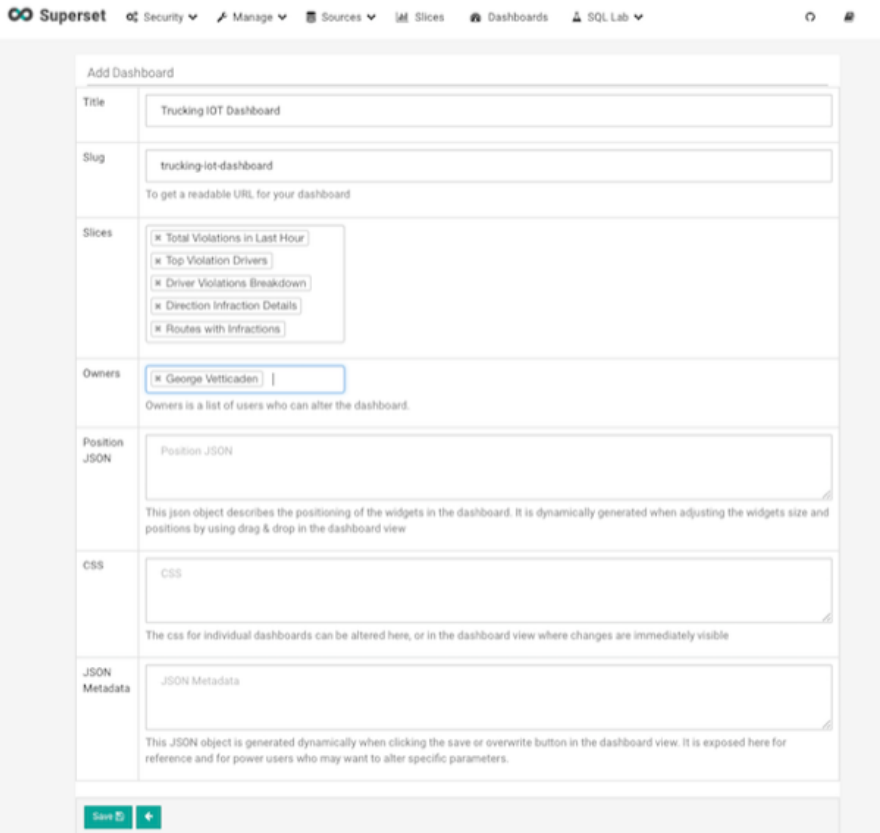
6. To save the slice, specify a name and name and click **Save**.



3.2. Adding Insight Slices to a Dashboard

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

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



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

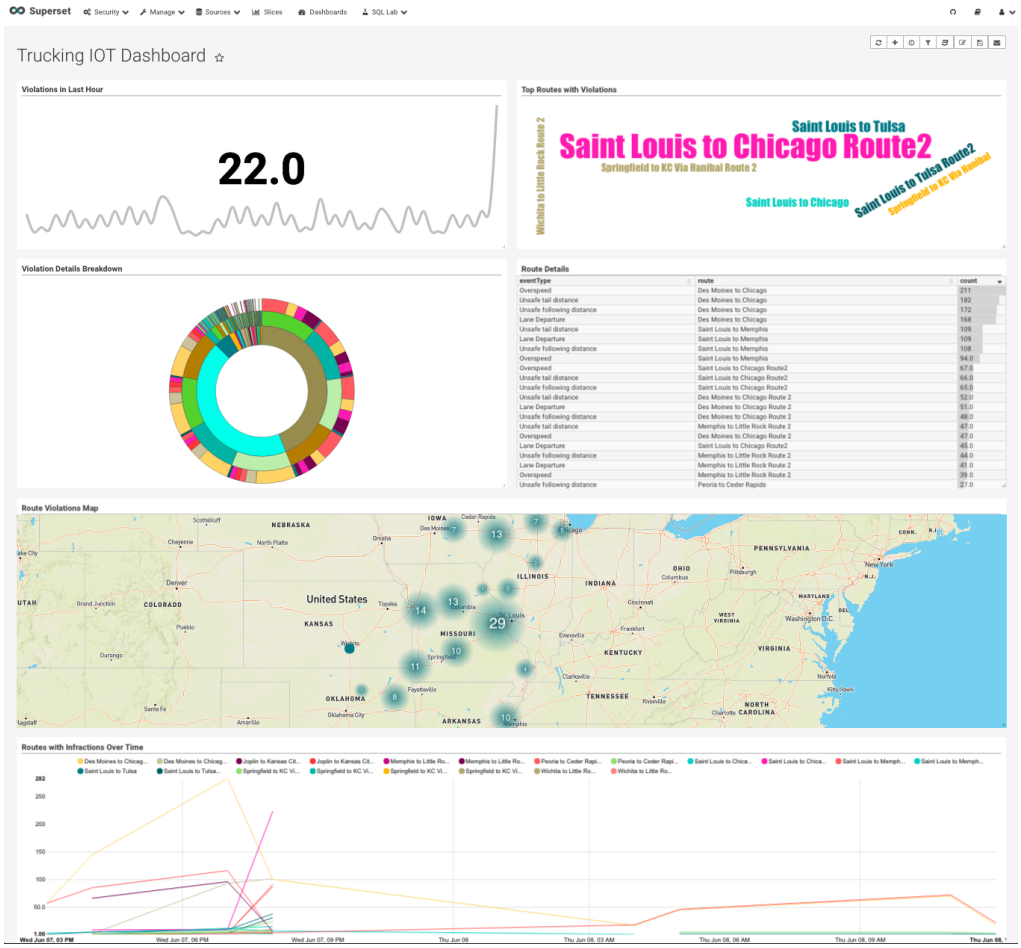
3.2.1. Dashboards for the Trucking IOT App

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

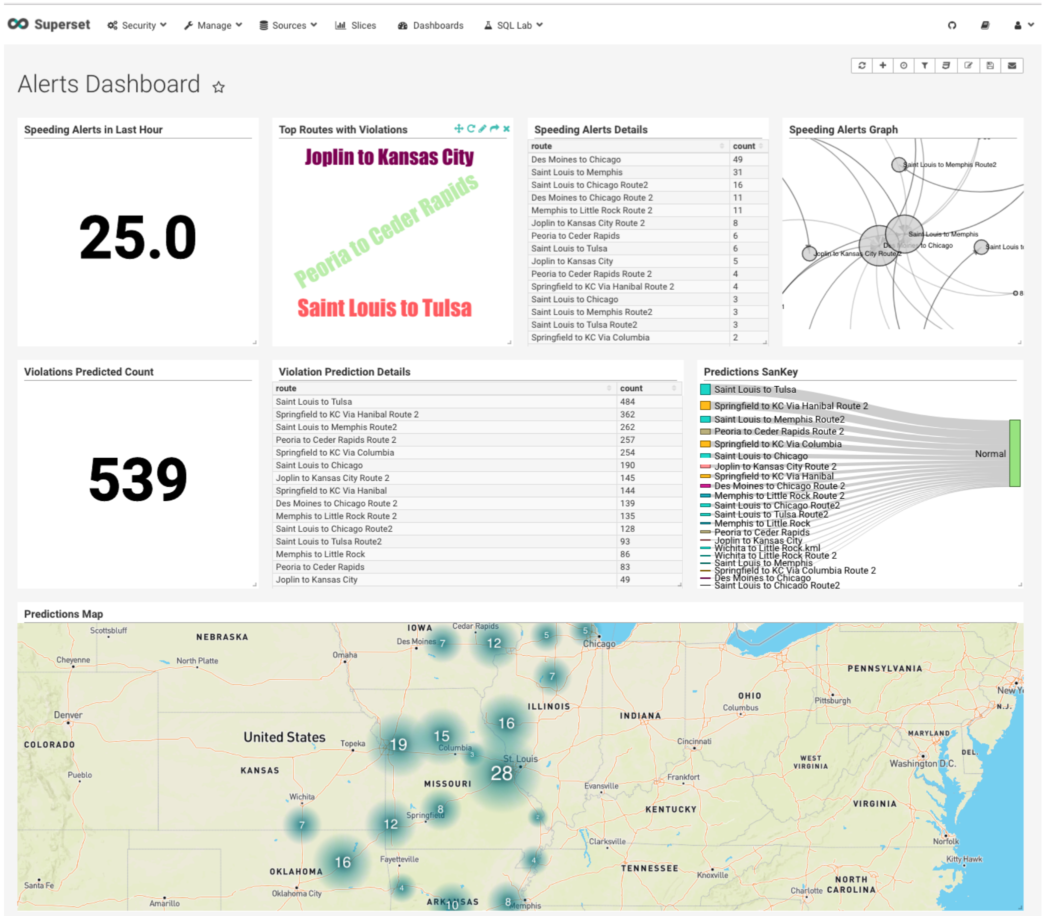
- violation-events-cube
- alerts-speeding-drivers-cube
- alerts-violation-predictions-cube

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

IoT Dashboard



Alerts Dashboard


Superset
Security
Manage
Sources
Slices
Dashboards
SQL Lab
Alerts Dashboard
Speeding Alerts in Last Hour
25.0
Top Routes with Violations
Joplin to Kansas City
Peoria to Ceder Rapids
Saint Louis to Tulsa
Speeding Alerts Details
route count
Des Moines to Chicago 49
Saint Louis to Memphis 31
Saint Louis to Chicago Route2 16
Des Moines to Chicago Route 2 11
Memphis to Little Rock Route 2 11
Joplin to Kansas City Route 2 8
Peoria to Ceder Rapids 6
Saint Louis to Tulsa 6
Joplin to Kansas City 5
Peoria to Ceder Rapids Route 2 4
Springfield to KC Via Hanibal Route 2 4
Saint Louis to Chicago 3
Saint Louis to Memphis Route2 3
Saint Louis to Tulsa Route2 3
Springfield to KC Via Columbia 2
Speeding Alerts Graph
Violations Predicted Count
539
Violation Prediction Details
route count
Saint Louis to Tulsa 484
Springfield to KC Via Hanibal Route 2 362
Saint Louis to Memphis Route2 262
Peoria to Ceder Rapids Route 2 257
Springfield to KC Via Columbia 254
Saint Louis to Chicago 190
Joplin to Kansas City Route 2 145
Springfield to KC Via Hanibal 144
Des Moines to Chicago Route 2 139
Memphis to Little Rock Route 2 135
Saint Louis to Chicago Route2 128
Saint Louis to Tulsa Route2 93
Memphis to Little Rock 86
Peoria to Ceder Rapids 83
Joplin to Kansas City 49
Predictions SanKey
Normal
Predictions Map
United States

4. Adding Custom Builder Components

You can use the SAM SDK to add custom components to your SAM applications.

4.1. Adding Custom Processors

To add custom processors to SAM, create the processors and then register it with SAM.

1. Createing Custom Processors
2. Registering Custom Processors
3. Creating a Custom Streaming Application

4.1.1. Creating Custom Processors

About This Task

Create a custom processor using the SDK, and package it into a jar file with all of its dependencies.

Steps

1. Create a new maven project using this maven pom file as an example.
2. To implement a custom processor, implement the following interface:

```
org.apache.streamline.streams.runtime.CustomProcessorRuntime
```

3. Package the jar file with all dependencies, by running the following commands:

```
mvn clean package
mvn assembly:assembly
```

4. In the target directory you should have an uber jar that ends with `jar-with-dependencies.jar`. You need this jar file when you register your custom processor with SAM.

Example

The PhoenixEnrichmentProcessor is a good example of a new custom processor implementation.

4.1.2. Registering Custom Processors with SAM

About This Task

You have to register each custom processor in SAM before you can use it for the first time.

Steps

1. From the left-hand SAM Global menu, hover over the **Configuration** menu, click **Application Resources**, and then click the **Custom Processor** tab.

2. Click the + icon to add a new processor.

3. Enter details for the custom processor.

Result

It might take a few minutes to upload the jar file to the server. Do not navigate away until you see a response. If you do not see a response, return to the Custom Processor page again; do not click Save again.

4.1.3. Creating a Custom Streaming Application

About This Task

After you have registered your custom processor, create a new stream application.

Steps

1. From **My Applications** click the + icon and launch the **Add Application** dialog.

2. Find your new processor in the **Processor Toolbar**, drag it onto the canvas, and configure it.

Result

When you double-click on your new custom processor, the configuration fields are exposed. Notice that the configuration is based on the "Config Fields" settings specified during the registration process.

4.2. Adding Custom Functions

User Defined Aggregate Functions (UDAF) allow you to add custom aggregate functions to SAM. Once you create and register UADFs they are available for use in the Aggregate processor.

User Defined Functions (UDFs) allow you to do simple transformations on event streams. This is used in the Projection processor.

This section provides information on how to create, build, and upload these custom functions.

1. Creating UDAFs

2. Creating UDFs

3. Building Custom Functions

4. Uploading Custom Functions to SAM

4.2.1. Creating UDAFs

About This Task

User Defined Aggregate Functions (UDAF) allow you to add custom aggregate functions to SAM. Once you create and register UADFs they are available for use in the Aggregate processor. Use these steps to create a new UADF.

Steps

1. Create a UADF by implement the following interface:

```
public interface UDAF<A, V, R> {
   A init();
   A add(A aggregate, V val);
   R result(A aggregate);
}
```

   Where:

   - A – Is the type of the aggregate that is used to aggregate the values. init returns the initial value for the aggregate.
   - V – is the type of the values we are processing. The add method is invoked with the current aggregate and the value for each of the events in the window. add is expected to aggregate the current value and return the updated aggregate.
   - R – is the result type and the result function takes the final aggregated value and returns the result.

2. For aggregate functions that requires two parameters, the UDAF2 interface also requires implementation. The only difference is that the add function is passed the current value of the aggregate and two values instead of one.

```
public interface UDAF2<A, V1, V2, R> {
   A init();
   A add(A aggregate, V1 val1, V2 val2);
   R result(A aggregate);
}
```

Example

In this example, you want to compute the average values of a particular field for events within a window. To do that, define an average aggregate function by implementing the UDAF interface as shown below:

```
// Here the aggregate is a pair that holds the running sum and the count of
 elements seen so far
// The values are integers and the result is a double.
public class MyAvg implements UDAF<Pair<Integer, Integer>, Integer, Double> {

// Here we initialize the aggregate and return its initial value (sum = 0 and
 count = 0).
@Override
public Pair<Integer, Integer> init() { return Pair.of(0, 0); }

// Here we update the sum and count values in the aggregate and return the
 updated aggregate
@Override
public Pair<Integer, Integer> add(Pair<Integer, Integer> agg, Integer val) {
       return Pair.of(agg.getKey() + val, agg.getValue() + 1);
```

```
   }

// Here we return the value of the sum divided by the count which is the
 average of the aggregated values.
   @Override
   public Double result(Pair<Integer, Integer> agg) {
       return (double) agg.getKey() / agg.getValue();
   }
}
```

4.2.2. Creating UDFs

About This Task

User Defined Functions (UDFs) allow you to do simple transformations on event streams. This is used in the Projection processor.

Steps

1. Create a UDF by implement the following interface:

   ```
   public interface UDF<O, I> {
      O evaluate(I i);
   }
   ```

   Where:

   - I – Is the input type.
   - O – Is the output type.
   - The evaluate method is invoked with the corresponding field value for each event in the stream.

2. For functions that accept two or more parameters, the there are corresponding UDF interfaces (UDF2 to UDF7).

   ```
   public interface UDF2<O, I1, I2> {
      O evaluate(I1 input1, I2 input2);
   }
   ```

Example 1

The ConvertToTimestampLong UDF is a good example of a new UDF implementation.

Example 2

In this example, you to concatenate the values of two fields of an event. To do this, define a MyConcat function by implementing the UDF2 interface as shown below

```
pubic class MyConcat implements UDF2<String, String, String> {
   public String evaluate(String s1, String s2) {
       return s1.concat(s2);
   }
}
```

4.2.3. Building Custom Functions

About This Task

Once you have created a UDAF, create a new maven project and build the `.jar` files to add to SAM. You can have multiple UDAFs in a single maven project. All of them are bundled into a single jar which can be uploaded.

Steps

1. Create a new maven project and add `streamline-sdk`. A sample `pom.xml` file is provided below.
2. Generate the UDAF `.jar` file by running:

```
mvn clean install
```

Result

The UDAF `.jar` file is created and you are ready to upload it to SAM.

Example pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
    <groupId>test</groupId>
    <version>0.1</version>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>my-custom-functions</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.hortonworks.streamline</groupId>
            <artifactId>streamline-sdk</artifactId>
            <version>0.1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

4.2.4. Uploading Custom Functions to SAM

About This Task

Once you have created and built the UDAF, upload it to SAM so that it is available in the Aggregate processor.

Steps

1. From the left-hand menu, select **Configuration**, then **Application Resources**.
2. Click the **UDF** tab. You use the UDF tab to handle both UDFs and UDAFs.

3. Click the **Add** icon to display the **Add UDF**.
4. Supply the following information, and click **Ok**.
   - Name – This is the internal name of the UDAF. This needs to be unique and should not conflict with any of the built in aggregate functions.
   - Display Name – This is what gets displayed in the list of aggregate functions in the Aggregate processor UI.
   - Description – This can be any textual description of the function to assist the user.
   - Type – This should be AGGREGATE for UDAFs, or FUNCTION for UDFs.
   - Classname – This is the full qualified class name of the UDAF that gets packaged in the Jar.
   - UDF JAR – Browse and select the jar file that you built using the maven project.

Result

Your new UDF or UDAF displays in the list of available functions.

5. 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

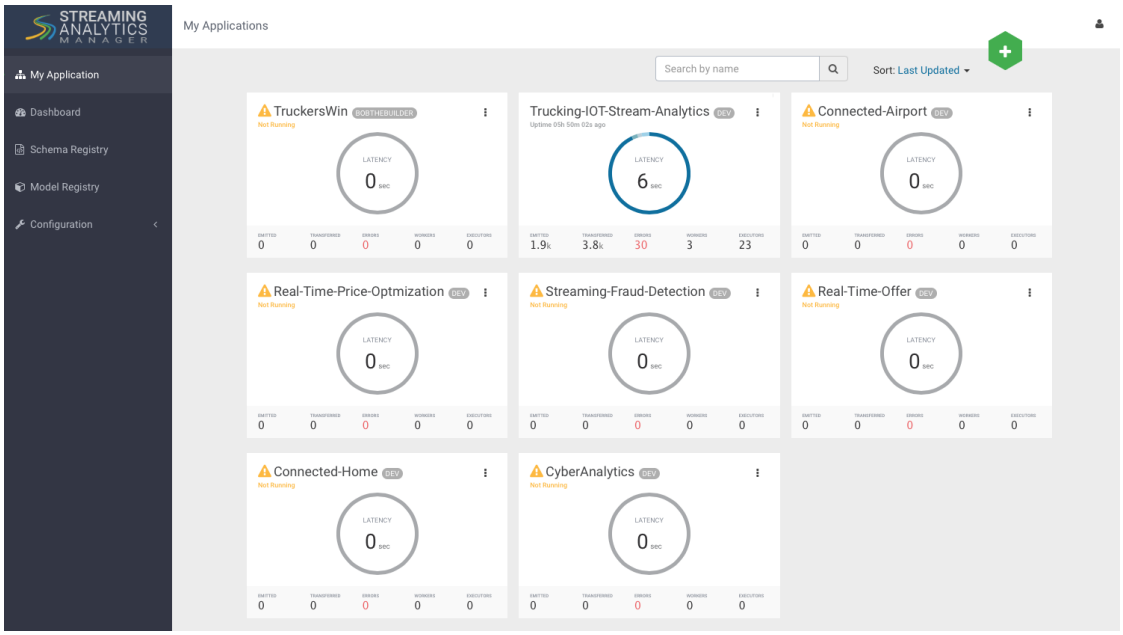
5.1. My Applications View

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

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

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

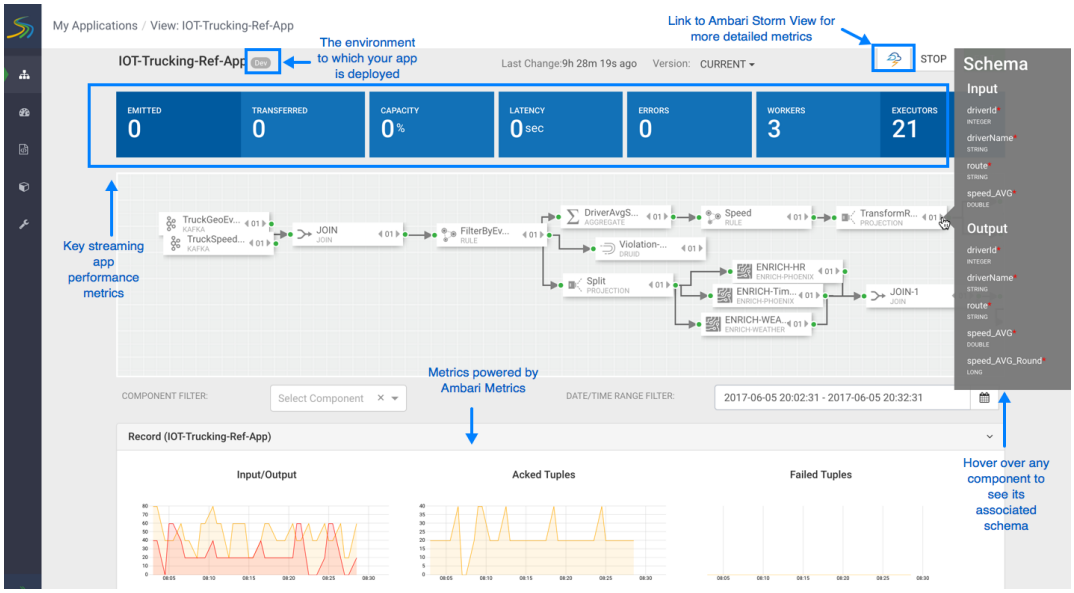
Each stream application is represented by an application tile. Hovering over the application tile provides status, metrics, and actions you can perform on the stream application.



5.2. Application Performance Monitoring

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

The following diagram describes elements of the APM view.



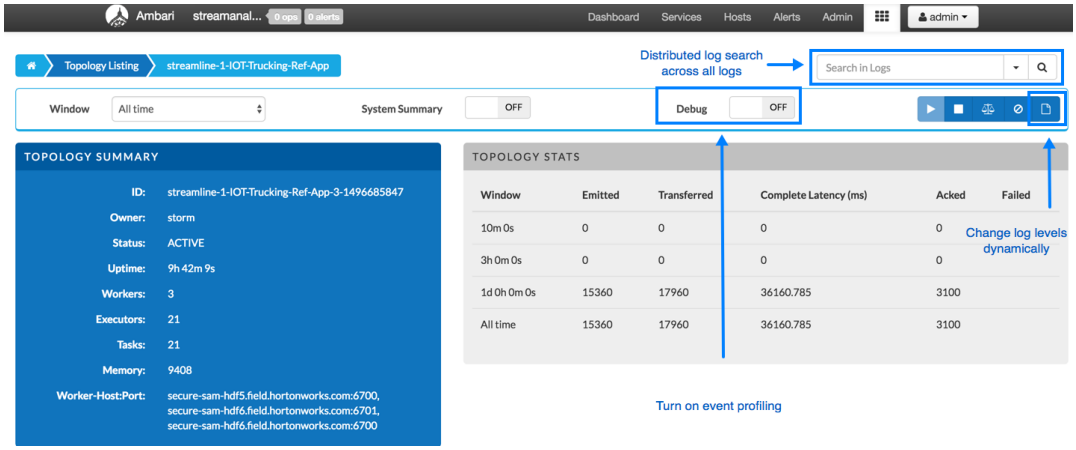
5.3. Troubleshooting and Debugging a Stream application

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

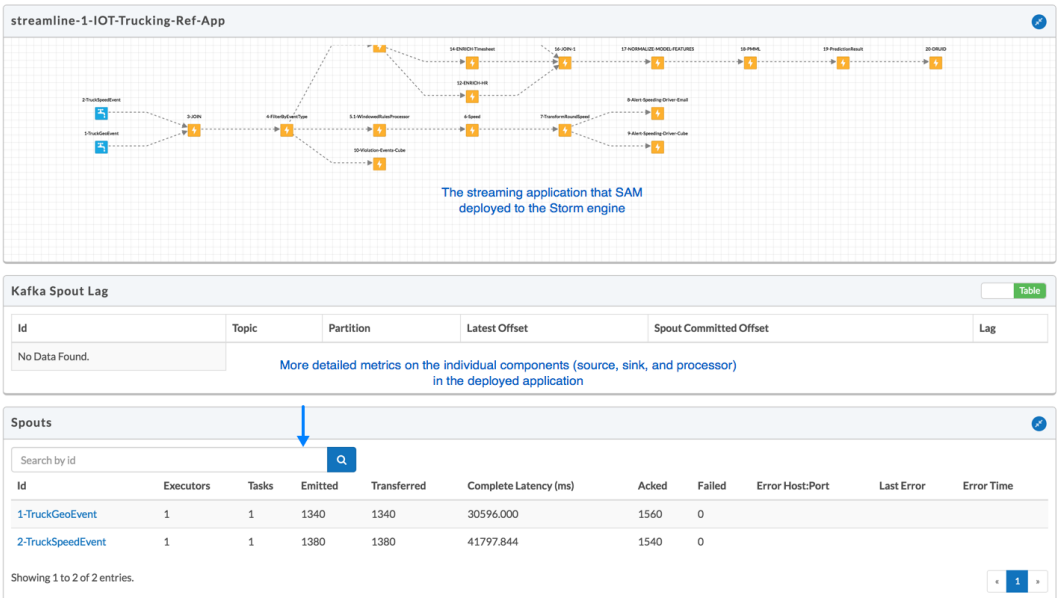
The Storm Ambari View provides the following capabilities for deeper troubleshooting and debugging:

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

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

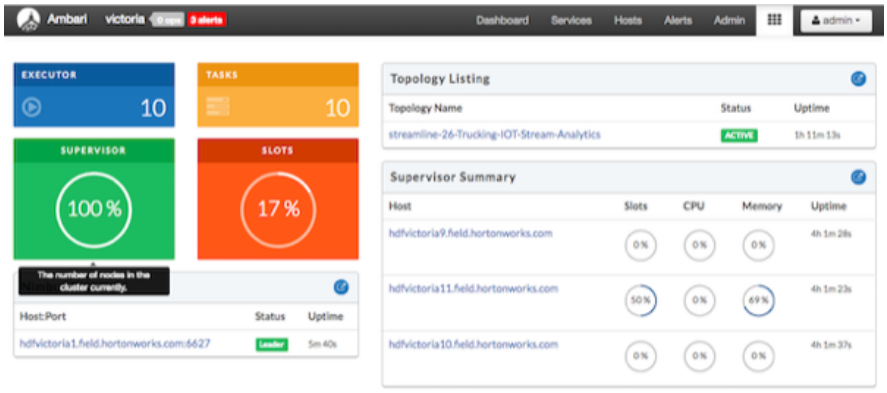


Scroll down to review the deployed topology and see metrics about its components.



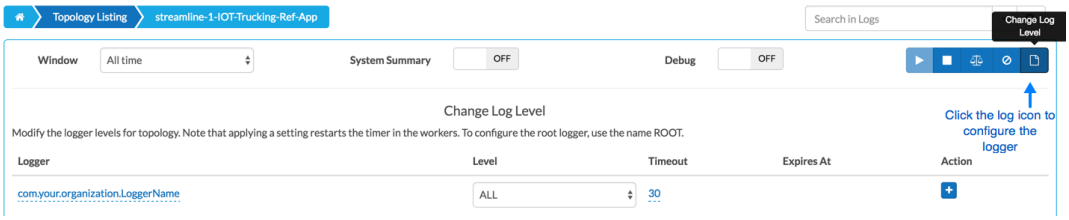
5.3.1. Streaming Engine Infrastructure Metrics

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



5.3.2. Changing Log Levels Dynamically and with Expiration Policies

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

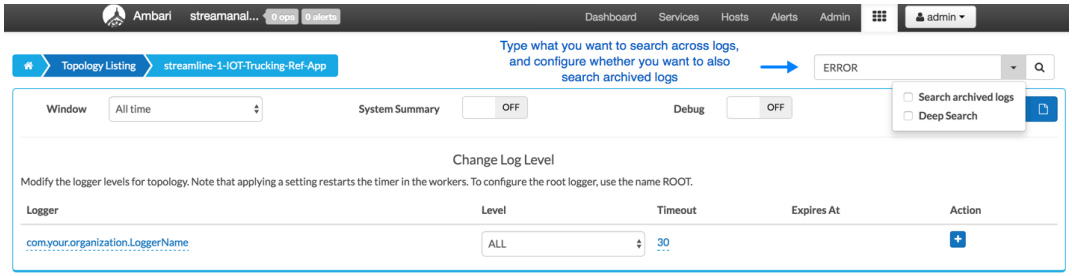


5.3.3. Distributed Log Search

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

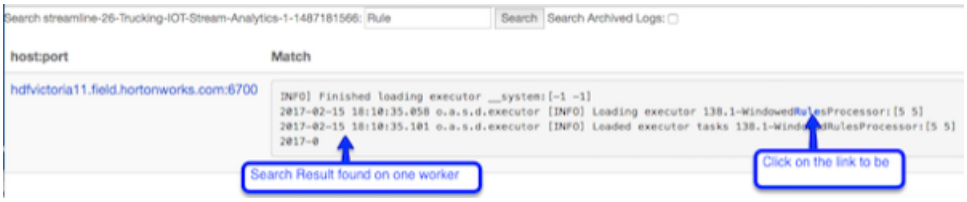
The following steps describe how to use distributed log search.

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



2. Review the results.

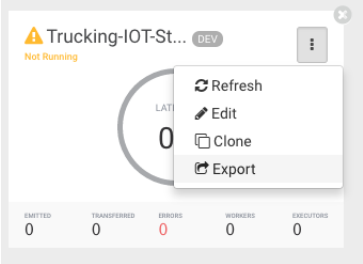
3. Click on the link to navigate to the exact location in the log file.



5.4. Exporting and Importing Stream applications

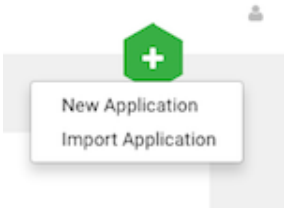
Service pool and environment abstractions combined with import and export capabilities allow you to move a stream application from one environment to another easily.

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

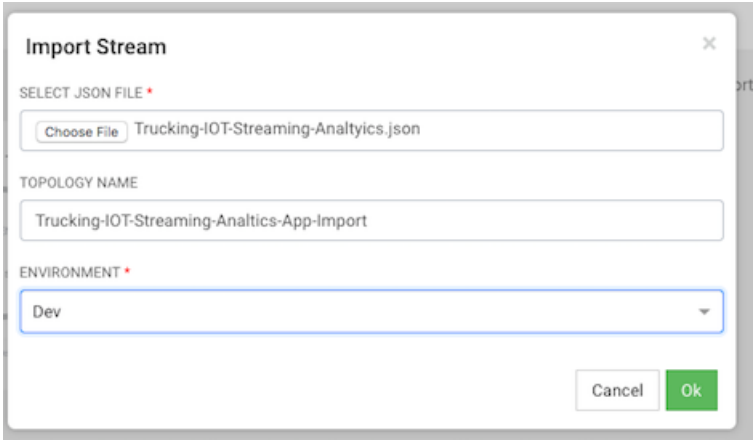


To import a stream application that was exported in JSON format:

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



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


Import Stream
SELECT JSON FILE *
Choose File
Trucking-IOT-Streaming-Analtyics.json
TOPOLOGY NAME
Trucking-IOT-Streaming-Analtics-App-Import
ENVIRONMENT *
Dev
Cancel
Ok

6. Source, Processor, and Sink Configuration Values

As you build your streaming applications, use this reference material to help configure the source, processor, and sink Stream Builder components.

6.1. Source Configuration Values

Table 6.1. Kafka

Configuration Field	Description, requirements, tips for configuration
Cluster Name	Mandatory. Service pool defined in SAM to get metadata information about Kafka cluster
Security Protocol	Mandatory. Protocol to be used to communicate with kafka brokers. E.g. PLAINTEXT. Auto suggest with a list of protocols supported by Kafka service based on cluster name selected. If you select a protocol with SSL or SASL make sure to fill out the related config fields
Bootstrap Servers	Mandatory. A comma separated string of host:port representing Kafka broker listeners. Auto suggest with a list of options based on security protocol selected above
Kafka topic	Mandatory. Kafka topic to read data from. Make sure that corresponding schema for topic is defined in Schema Registry
Consumer Group Id	Mandatory. A unique string that identifies the consumer group it belongs to. Used to keep track of consumer offsets
Reader schema version	Optional. Version of schema for topic to read from. Default value is the version used by producer to write data to topic
Kerberos client principal	Optional(Mandatory for SASL). Client principal to use to connect to brokers while using SASL GSSAPI mechanism for Kerberos(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Kerberos keytab file	Optional(Mandatory for SASL). Keytab file location on worker node containing the secret key for client principal while using SASL GSSAPI mechanism for Kerberos(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Kafka service name	Optional(Mandatory for SASL). Service name that Kafka broker is running as(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Fetch minimum bytes	Optional. The minimum number of bytes the broker should return for a fetch request. Default value is 1
Maximum fetch bytes per partition	Optional. The maximum amount of data per-partition the broker will return. Default value is 1048576
Maximum records per poll	Optional. The maximum number of records a poll will return. Default value is 500

Poll timeout(ms)	Optional. Time in milliseconds spent waiting in poll if data is not available. Default value is 200
Offset commit period(ms)	Optional. Period in milliseconds at which offsets are committed. Default value is 30000
Maximum uncommitted offsets	Optional.Defines the max number of polled records that can be pending commit, before another poll can take place. Default value is 10000000. This value should depend on the size of each message in Kafka and the memory available to the worker jvm process
First poll offset strategy	Optional. Offset used by the Kafka spout in the first poll to Kafka broker. Pick one from enum values. ["EARLIEST", "LATEST", "UNCOMMITTED_EARLIEST", "UNCOMMITTED_LATEST"]. Default value is EARLIEST_UNCOMMITTED. It means that by default it will start from the earliest uncommitted offset for the consumer group id provided above
Partition refresh period(ms)	Optional. Period in milliseconds at which Kafka will be polled for new topics and/or partitions. Default value is 2000
Emit null tuples?	Optional. A flag to indicate if null tuples should be emitted to downstream components or not. Default value is false
First retry delay(ms)	Optional. Interval delay in milliseconds for first retry for a failed Kafka spout message. Default value is 0
Retry delay period(ms)	Optional. Retry delay period(geometric progression) in milliseconds for second and subsequent retries for a failed Kafka spout message. Default value is 2
Maximum retries	Optional. Maximum number of times a failed message is retried before it is acked and committed. Default value is 2147483647
Maximum retry delay(ms)	Optional. Maximum interval in milliseconds to wait before successive retries for a failed Kafka spout message. Default value is 10000
Consumer startup delay(ms)	Optional. Delay in milliseconds after which Kafka will be polled for records. This value is to make sure all executors come up before first poll from each executor happens so that partitions are well balanced among executors and onPartitionsRevoked and onPartitionsAssigned is not called later causing duplicate tuples to be emitted. Default value is 60000
SSL keystore location	Optional.The location of the key store file. Used when Kafka client connectivity is over SSL
SSL keystore location	Optional. The store password for the key store file
SSL key password	Optional. The password of the private key in the key store file
SSL truststore location	Optional(Mandatory for SSL). The location of the trust store file
SSL truststore password	Optional(Mandatory for SSL). The password for the trust store file
SSL enabled protocols	Optional. Comma separated list of protocols enabled for SSL connections
SSL keystore type	Optional. File format of keystore file. Default value is JKS
SSL truststore type	Optional. File format of truststore file. Default value is JKS
SSL protocol	Optional. SSL protocol used to generate SSLContext. Default value is TLS
SSL provider	Optional. Security provider used for SSL connections. Default value is default security provider for JVM

SSL cipher suites	Optional. Comma separated list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported
SSL endpoint identification algorithm	Optional. The endpoint identification algorithm to validate server hostname using server certificate
SSL key manager algorithm	Optional. The algorithm used by key manager factory for SSL connections. Default value is SunX509
SSL secure random implementation	Optional. The SecureRandom PRNG implementation to use for SSL cryptographic operations
SSL trust manager algorithm	Optional. The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine. Default value is PKIX

Table 6.2. Event Hubs

Configuration Field	Description, requirements, tips for configuration
Username	The Event Hubs user name (policy name in Event Hubs Portal)
Password	The Event Hubs password (shared access key in Event Hubs Portal)
Namespace	The Event Hubs namespace
Entity Path	The Event Hubs entity path
Partition Count	The number of partitions in the Event Hubs
ZooKeeper Connection String	The ZooKeeper connection string
Checkpoint Interval	The frequency at which offsets are checkpointed
Receiver Credits	Receiver credits
Max Pending Messages Per Partition	The max pending messages per partition
Enqueue Time Filter	The enqueue time filter
Consumer Group Name	The consumer group name

Table 6.3. HDFS

Configuration Field	Description, requirements, tips for configuration
Cluster Name	Service pool defined in SAM to get metadata information about HDFS cluster
HDFS URL	HDFS namenode URL
Input File Format	The format of the file being consumed dictates the type of reader used to read the file. Currently only 'com.hortonworks.streamline.streams.runtime.storm.spout.JsonFileReader' is supported
Source Dir	The HDFS directory from which to read the files.
Archive Dir	Files from source dir will be moved to this HDFS location after being completely read.
Bad Files Dir	Files from Source Dir will be moved to this HDFS location if there is an error encountered while processing them.
Lock Dir	Lock files (used to synchronize multiple reader instances) will be created in this location. Defaults to a '.lock' subdirectory under the source directory.
Commit Frequency Count	Records progress in the lock file after specified number of records are processed. Setting it to 0 disables this.

Commit Frequency Secs	Records progress in the lock file after specified secs have elapsed. Must be greater than 0.
Max Outstanding	Limits the number of unACKed tuples by pausing tuple generation (if ACKers are used in the topology).
Lock Timeout Seconds	Duration of inactivity after which a lock file is considered to be abandoned and ready for another spout to take ownership.
Ignore Suffix	File names with this suffix in the source dir will not be processed.

6.2. Processor Configuration Values

Table 6.4. Aggregate

Configuration Field	Description, requirements, tips for configuration
General Processor description	Performs aggregate operations on a stream of events within a window.
Select Keys	These are the keys to “group by” for computing the aggregate.
Window Interval Type	Time - for time based windows. Count - for count based windows.
Window Interval	The length or duration of the window
Sliding Interval	The interval at which the window slides
Timestamp Field	A field in the event that represents the event timestamp as a long value. If specified the timestamp at which the event occurred will be used for the window computations.
Output Fields – Input	The field on which to apply aggregate function
Output Fields – Aggregate Function	The aggregate function to apply
Output Fields – Output	The output field name

Table 6.5. Branch

Configuration Field	Description, requirements, tips for configuration
General processor description	Conditionally redirects tuples from one incoming stream to one or more outbound streams.
Process all checkbox	If disabled, stops processing further rules after a rule evaluates successfully.
Rule Name	Rule name. Must be unique within the Branch processor.
Rule Description	Description of rule
Field Name	Field name used in the condition for the rule
Rule Operation	The comparison operator for the condition

Table 6.6. Join

Configuration Field	Description, requirements, tips for configuration
General Processor Description	Joins one or more event streams into one output stream, based on user defined join criteria
Select Stream	Name of stream to join
Select Field	Name of field to use for join
Window Interval Type	Determines the type of windowing (count/time based) to use for buffering streams to be joined
Window Interval	The window size.

Sliding Interval	The interval between the start of two consecutive windows
Output Fields	Select which of the fields to include in the resulting event

Table 6.7. PMML

Configuration Field	Description, requirements, tips for configuration
General Processor Description	Allows users to score tuples according to a choice of PMML model registered in the model registry. The scored results are put in the predicted fields as defined in the PMML XML descriptor file. Predicted fields are available to send downstream, in addition to input fields
Model Name	Name of the PPML model in model registry to use

Table 6.8. Projection Bolt

Configuration Field	Description, requirements, tips for configuration
General Processor Description	This allows user to choose specific fields from the input events to be passed to output event and apply a transformation using UDF on chosen fields and add result as a field in the output event.
Projection Fields	Input event fields to be projected into output event.
Function	UDF to be applied on the given input fields and output is added as a new field in the output event.
Arguments	Field names to be passed as arguments to the chosen function
Fields Name	Name of the inu
Plus icon	Add a new transformation

Table 6.9. Rule

Configuration Field	Description, requirements, tips for configuration
General Processor Description	Design time definition of a rule whose scope is the input fields. The condition of the rule is defined in the Create Query section. Only runtime values whose rule condition evaluates to true will be sent downstream.
Rule Name	Name of the rule. It must be unique only within a Rule processor. Can be reused across rule processors.
Description	Documentation detailing the purpose of the rule. For user reference only.
Create Query	The condition of the rule is a composition of boolean expressions built with operators on input fields. These boolean expressions are parsed as SQL like query.

6.3. Sink Configuration Values

Table 6.10. Cassandra

Configuration Field	Description, requirements, tips for configuration
General Sink Description	This allows users to send events into given cassandra table.
Table Name	Name of the table into which events should be written to.
Column Name	Column name to which a respective field is mapped.
Field Name	Field name to be mapped as respective column name.
Cassandra Configurations- User Name	User name to connect to Cassandra cluster.

Password	Password to connect to Cassandra cluster.
Keyspace	Keyspace in which table exists
Nodes	Cassandra nodes configuration to be passed
Port	Port number for Cassandra cluster
Row Batch Size	Maximum number of rows to be taken in a batch
Retry Policy	Class name of the retry policy to be applied. Default value is “DefaultRetryPolicy”. Valid options are "DowngradingConsistencyRetryPolicy", "FallthroughRetryPolicy" and "DefaultRetryPolicy"
Consistency Level	Consistency level at which data is inserted. Default value is: QUORUM, valid values are ["ANY", "ONE", "TWO", "THREE", "QUORUM", "ALL", "LOCAL_QUORUM", "EACH_QUORUM", "SERIAL", "LOCAL_SERIAL", "LOCAL_ONE"]
Reconnection Base Delay	Base delay (in milliseconds) while reconnecting to target.
Reconnection Maximum Delay	Maximum delay (in milliseconds) while reconnecting to target.

Table 6.11. Druid

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Druid sink is used to push data Druid data store. This sink uses Druid’s Tranquility library to push data. More details : http://druid.io/docs/latest/ingestion/stream-push.html
Name of the Indexing Service	The druid.service name of the indexing service overlord node. It is mandatory parameter.
Service Discovery path	Curator service discovery path. It is mandatory parameter.
ZooKeeper Connect String	ZooKeeper connect string. It is mandatory parameter.
Datasource name	The name of the ingested data source. Datasources can be thought of as tables. It is mandatory parameter.
Dimensions	Specifies the dimensions(columns) of the data. It is mandatory parameter.
TimeStamp Field Name	Specifies the column and format of the timestamp.It is mandatory parameter.
Window Period	Window Period takes ISO 8601 Period format (https://en.wikipedia.org/wiki/ISO_8601). It is mandatory parameter.
Index Retry Period	If an indexing service overlord call fails for some apparently-transient reason, retry for this long before giving up. It takes ISO 8601 Period format (https://en.wikipedia.org/wiki/ISO_8601). It is mandatory parameter.
Segment Granularity	The granularity to create segments.
Query Granularity	The minimum granularity to be able to query results at and the granularity of the data inside the segment.
Batch Size	Maximum number of messages to send at once
Max Pending Batches	Maximum number of batches that may be in flight
Linger millis	Wait this long for batches to collect more messages (up to maxBatchSize) before sending them.
Block On Full	Whether send will block (true) or throw an exception (false) when called while the outgoing queue is full
Druid partitions	Number of Druid partitions to create.
Partition Replication	Number of instances of each Druid partition to create.

Aggregator Info	A list of aggregators. Currently we support Count Aggregator, Double Sum Aggregator, Double Max Aggregator, Double Min Aggregator, Long Sum Aggregator, Long Max Aggregator, Long Min Aggregators.

Table 6.12. Hive

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Hive sink is used to write data to Hive tables
Metastore URI	URI of the metastore to connect to eg: thrift://localhost:9083
Database Name	Name of the Hive database
Table name	Name of table to stream to
Fields	The event fields to stream to hive
Partition fields	The event fields on which to partition the data
Flush Interval	The interval (in seconds) at which a transaction batch is committed
Transactions per batch	The number of transactions per batch
Max open connections	The maximum number of open connections to Hive
Batch size	The number of events per batch
Idle timeout	The idle timeout
Call timeout	The call timeout
Heartbeat Interval	The heart beat interval
Auto create partitions	If true, the partition specified in the endpoint will be auto created if it does not exist
Kerberos keytab	Kerberos keytab file path
Kerberos principal	Kerberos principal name

Table 6.13. HBase

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Writes to events to HBase
HBase table	Hbase table to write to
Column Family	Hbase table column family
Batch Size	Number of records in the batch to trigger flushing. Note that every batch needs to be full before it can be flushed as tick tuple is not supported currently due to the fact that all bolts in topology receive a tick tuple if enabled
Row Key Field	Field to be used as row key for table

Table 6.14. HDFS

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Writes events to HDFS
Hdfs URL	Hdfs Namenode URL
Path	Directory to which the files will be written
Flush Count	Number of records to wait for before flushing to Hdfs
Rotation Policy	Strategy to rotate files in Hdfs
Rotation Interval Multiplier	Rotation interval multiplier for timed rotation policy
Rotation Interval Unit	Rotation interval unit for timed rotation policy
Output fields	Specify the output fields, in the desired order

Prefix	Prefix for default file name format
Extension	Extension for default file name format

Table 6.15. JDBC

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Writes events to a database using JDBC.
Driver Class Name	The driver class name. E.g. com.mysql.jdbc.Driver
JDBC URL	JDBC Url, E.g. jdbc:mysql://localhost:3306/test
User Name	Database username.
Password	Database password.
Table Name	Table to write to.
Column Names	Names of the database columns

Table 6.16. Kafka

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Kafka sink to write SAM events to a kafka topic
Cluster Name	Mandatory. Service pool defined in SAM to get metadata information about Kafka cluster
Kafka Topic	Mandatory. Kafka topic to write data to. Make sure that the schema for the corresponding topic exists in SR. The incoming SAM event into Kafka sink should adhere to the version of schema selected
Security Protocol	Mandatory. Protocol to be used to communicate with kafka brokers. E.g. PLAINTEXT. Auto suggest with a list of protocols supported by Kafka service based on cluster name selected. If you select a protocol with SSL or SASL make sure to fill out the related config fields
Bootstrap Servers	Mandatory. A comma separated string of host:port representing Kafka broker listeners. Auto suggest with a list of options based on security protocol selected above
Fire And Forget?	Optional. A flag to indicate if kafka producer should wait for ack or not. Default value is false
Async?	Optional. A flag to indicate whether to use async kafka producer or not. Default value is true
Key serializer	Optional. Type of key serializer to use. Options are ["String", "Integer", "Long", "ByteArray"]. Default value is ByteArray. Note that this field does not save any key in the kafka message. Incoming SAM event is stored as value in Kafka message with key being null
Key field	Optional. Name of the key field. One of the fields from incoming event schema
Writer schema version	Optional. Version of schema for topic to use for serializing the message. Default is the latest version for the schema
Ack mode	Optional. Ack mode used in producer request for a record sent to server(None|Leader|Min in-sync replicas). Options are [“None”, “Leader”, “All”]. Default value is “Leader”
Buffer memory	Optional. The total bytes of memory the producer can use to buffer records waiting to be sent to the server. Default value is 33554432
Compression type	Optional. The compression type for all data generated by the producer. Options are ["none", "gzip", "snappy", "lz4"]. Default value is “none”

Retries	Optional. Number of retry attempts for a record send failure. Default value is 0
Batch size	Optional. Producer batch size in bytes for records sent to same partition. Default value is 16384
Client id	Optional. Id sent to server in producer request for tracking in server logs
Max connection idle	Optional. Time in milliseconds for which connections can be idle before getting closed. Default value is 540000
Linger time	Optional. Time in milliseconds to wait before sending a record out when batch is not full. Default value is 0
Max block	Optional. Time in milliseconds that send and partitionsFor methods will block for. Default value is 60000
Max request size	Optional. Maximum size of a request in bytes. Default value is 1048576
Receive buffer size	Optional. Size in bytes of TCP receive buffer (SO_RCVBUF) to use when reading data. Default value is 32768
Request timeout	Optional. Maximum amount of time in milliseconds the producer will wait for the response of a request. Default value is 30000
Kerberos client principal	Optional(Mandatory for SASL). Client principal to use to connect to brokers while using SASL GSSAPI mechanism for Kerberos(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Kerberos keytab file	Optional(Mandatory for SASL). Keytab file location on worker node containing the secret key for client principal while using SASL GSSAPI mechanism for Kerberos(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Kafka service name	Optional(Mandatory for SASL). Service name that Kafka broker is running as(used in case of security protocol being SASL_PLAINTEXT or SASL_SSL)
Send buffer size	Optional.Size in bytes of TCP send buffer (SO_SNDBUF) to use when sending data. Default value is 131072
Timeout	Optional. Maximum amount of time in milliseconds server will wait for acks from followers. Default value is 30000
Block on buffer full?	Optional. Boolean to indicate whether to block on a full buffer or throw an exception.Default value is true
Max in-flight requests	Optional. Maximum number of unacknowledged requests producer will send per connection before blocking. Default value is 5
Metadata fetch timeout	Optional. Timeout in milliseconds for a topic metadata fetch request. Default value is 60000
Metadata max age	Optional. Time in milliseconds after which a metadata fetch request is forced. Default value is 300000
Reconnect backoff	Optional. Amount of time in milliseconds to wait before attempting to reconnect to a host. Default value is 50
Retry backoff	Optional. Amount of time in milliseconds to wait before attempting to retry a failed fetch request. Default value is 100
SSL keystore location	Optional.The location of the key store file. Used when Kafka client connectivity is over SSL
SSL keystore location	Optional. The store password for the key store file
SSL key password	Optional. The password of the private key in the key store file
SSL truststore location	Optional(Mandatory for SSL). The location of the trust store file

SSL truststore password	Optional(Mandatory for SSL). The password for the trust store file
SSL enabled protocols	Optional. Comma separated list of protocols enabled for SSL connections
SSL keystore type	Optional. File format of keystore file. Default value is JKS
SSL truststore type	Optional. File format of truststore file. Default value is JKS
SSL protocol	Optional. SSL protocol used to generate SSLContext. Default value is TLS
SSL provider	Optional. Security provider used for SSL connections. Default value is default security provider for JVM
SSL cipher suites	Optional. Comma separated list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported
SSL endpoint identification algorithm	Optional. The endpoint identification algorithm to validate server hostname using server certificate
SSL key manager algorithm	Optional. The algorithm used by key manager factory for SSL connections. Default value is SunX509
SSL secure random implementation	Optional. The SecureRandom PRNG implementation to use for SSL cryptographic operations
SSL trust manager algorithm	Optional. The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine. Default value is PKIX

Table 6.17. Notification

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Can be used to send out notifications (currently supports email)
Username	The username for the mail server
Password	The password for the mail server
Host	Mail server host name
Port	Mail server port
SSL?	If the connection should be over SSL
Start TLS	Flag to indicate the TLS setting
Debug?	Whether to log debug messages
Email Server Protocol	The email server protocol. E.g. smtp
Authenticate	Flag to indicate if authentication is to be performed

Table 6.18. Open TSDB

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Sink to which events can be written given OpenTSDB cluster.
REST API URL	The URL of the REST API (ex: http://localhost:4242)
Metric Field Name	Field name of the metric
Timestamp Field Name	Field name of the timestamp
Tags Field Name	Field name of tags.
Value Field Name	Field name of the value
Fail Tuple for Failed Metrics?	Whether to fail tuple for any failed metrics to OpenTSDB

Sync?	Flag to indicate whether to sync or not.
Sync Timeout	Sync timeout in (milliseconds), this is taken into account only when Sync is true.
Return Summary?	Whether to return summary or not
Return Details?	Whether to return details or not.
Enable Chunked Encoding?	Whether to enable chunked encoding or not for REST API calls to OpenTSDB

Table 6.19. Solr

Configuration Field	Description, requirements, tips for configuration
General Sink Description	Enables indexing of live input data into Apache Solr collections
Apache Solr ZooKeeper Host String	Info about the zookeeper ensemble used to coordinate the Solr cluster. This string is specified in a comma separated value as folows: zk1.host.com:2181,zk2.host.com:2181,zk3.example.com:2181
Apache Solr Collection Name	The name of the Apache Solr collection where to index live data
Commit Batch Size	Defines how often the indexed data is committed into Apache Solr. It is specified using an integral number. For instance, if set to 100, every 100 tuples Apache Solr will commit the data