# Oracle® Cloud

# Using Stream Analytics in Oracle Integration Classic

E89784-07
January 2019

Oracle Cloud Using Stream Analytics in Oracle Integration Classic,

E89784-07



Primary Author: Oracle Corporation

# Contents

## 2 Administer Stream Analytics



## 3 Work with Stream Analytics Artifacts



## 4 Work with Patterns

## 5 Expression Builder Functions

# Preface

*User's Guide* describes how to get started with the product, how to build a simple pipeline, and how to build pipelines for specific use cases.

**Topics:**



## Audience

This document is intended for users of Stream Analytics, provisioned as part of the Integration Analytics service type.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

## Related Resources

See these Oracle resources:

- Oracle Cloud at `http://cloud.oracle.com`.
- Oracle Integration documentation in the Oracle Cloud Library on the Oracle Help Center.

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
| --- | --- |
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

1

# Get Started with Stream Analytics

Stream Analytics allows for the creation of custom operational dashboards that provide real-time monitoring and analyses of event streams in an Apache Spark based system. Stream Analytics enables customers to identify events of interest in their Apache Spark based system, execute queries against those event streams in real time and drive operational dashboards or raise alerts based on that analysis. Stream Analytics runs as a set of native Spark pipelines.

**Topics:**



## About Stream Analytics

Stream Analytics is an in-memory technology for real-time analytic computations on streaming data. The streaming data can originate from IoT sensors, web pipelines, log files, point-of-sale devices, ATM machines, social media, or from any other data source. Stream Analytics is available as a managed service in Oracle Cloud and as an on premises installation.

Stream Analytics is used to identify business threats and opportunities by filtering, aggregating, correlating, and analyzing high volume of data in real time.

Once a situation or alert is detected, appropriate action can be taken by triggering a business workflow using Oracle Integration or by sending results to a presentation service such as Oracle Analytics Cloud.

More precisely, Stream Analytics can be used in the following scenarios:

- Build complex event processing pipelines by blending and transforming data from disparate transactional and non-transactional sources.
- Perform temporal analytics based on time and event windows.
- Perform location-based analytics using built-in spatial patterns.
- Detect patterns in time-series data and execute real-time actions.
- Build operational dashboards by visualizing processed data streams.
- Use Machine Learning to score current event and predict next event.
- Run ad-hoc queries on results of processed data streams.

Some industry specific examples include:

- Detecting real-time fraud based on incoming transaction data.
- Tracking transaction losses and margins in real-time to renegotiate with vendors and suppliers.
- Improving asset maintenance by tracking healthy operating parameters and pro-actively scheduling maintenance.
- Improving margins by continuously tracking demand and optimizing markdowns instead of randomly lowering prices.
- Readjusting prices by continuously tracking demand, inventory levels, and product sentiment on social media etc.
- Marketing and making real-time offers based on customer location and loyalty.
- Instantly identifying shopping cart defections and improving conversion rates.
- Upselling products and services by instantly identifying customer's presence on company website.
- Improving asset utilization by tracking average time it takes to load and unload merchandise.
- Improving turnaround time by preparing dock and staff based on estimated arrival time of fleet.
- Revising schedule estimates based on actual time to enter and exit loading zones, and so on.

# Why Stream Analytics?

Various reasons and advantages encourage you to use Stream Analytics instead of similar products available in the industry.

### Oracle Managed

There is no infrastructure to worry and you can get started in minutes with Stream Analytics. Scale out when your data volume and velocity increases without worrying about virtual machines, cluster management, and so on.

### Simplicity

Author powerful data processing pipeliness using self-service web-based tool in Stream Analytics. The tool automatically generates a Spark pipeline along with instant visual validation of pipeline logic.

### Built on Apache Spark

Stream Analytics can attach to any version-compliant Yarn cluster running Spark and is first in the industry to bring event-by-event processing to Spark Streaming.

### Enterprise Grade

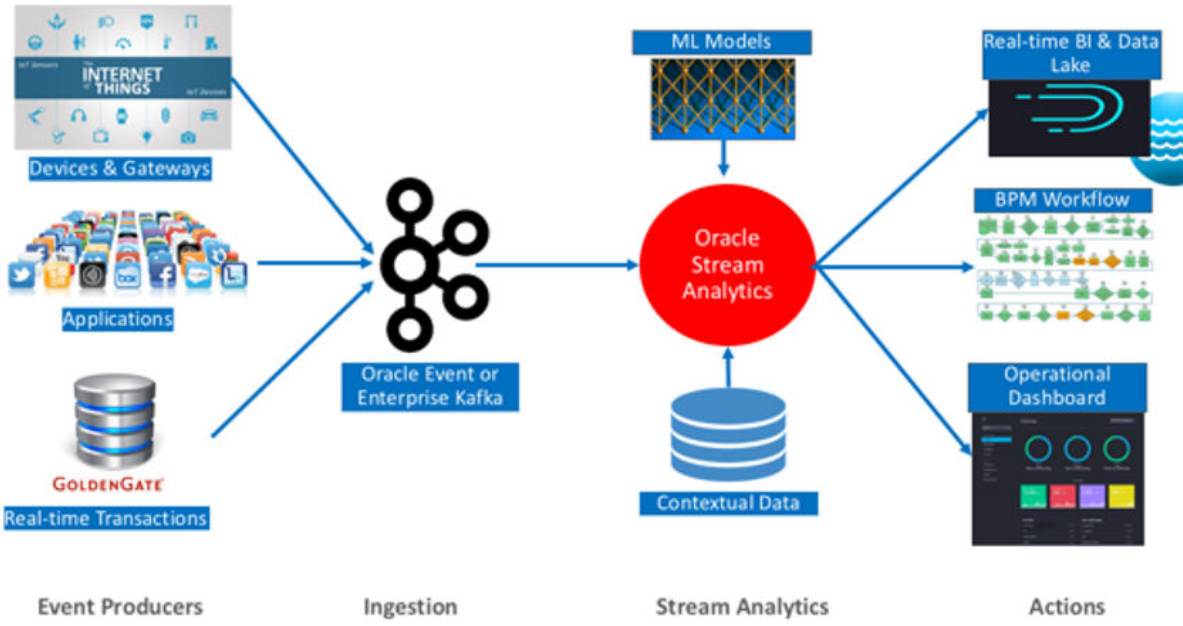
Stream Analytics is built on Apache Spark to provide full horizontal scale out and 24x7 availability of mission-critical workloads. Automated check-pointing ensures exact-once processing and zero data loss. Built-in governance provides full accountability of who did what and when to the system. As part of management and monitoring, Stream Analytics provides a visual representation of pipeline topology/relationships along with dataflow metrics to indicate number of events ingested, events dropped, and throughput of each pipeline.

# How Does Stream Analytics Work?

Stream Analytics starts with ingesting data from Kafka with first-class support for GoldenGate change data capture. Examining and analyzing the stream is performed by creating data pipelines.

A data pipeline can query data using time windows, look for patterns, and apply conditional logic while the data is still in motion. The query language used in Stream Analytics is called Continuous Query Language (CQL) and is similar to SQL. But CQL includes additional constructs for pattern matching and recognition. Though CQL is declarative, there is no need to write any code in Stream Analytics. The web-based tool automatically generates queries and the Spark Streaming pipeline. Once data is analyzed and situation is detected, the pipeline can terminate to trigger BPM workflows in Oracle Integration or save results into a Data Lake for deeper insights and intelligence using Oracle Analytics Cloud.

The following diagram illustrates the architecture of Stream Analytics:




The analyzed data is used to build operational dashboards, trigger workflows, and it is saved to Data Lakes for business intelligence and ad-hoc queries.

# Before You Begin with Stream Analytics

In Stream Analytics you typically start your work in the Catalog. The Catalog is a repository of entities that can be created by the user. There are different types of entities and in this section we are going to introduce each type one by one.

Every entity in the Catalog has a common subset of properties: a name, a description and several (zero, one or more) tags assigned to it. The tags assigned to an entity can later be used to filter the list of entities in the Catalog. In other words, tags can be used to arrange the entities into logical groups, such as projects by assigning the project name to the entity.

Entities may have one or more additional properties based on the entity's type and purpose. The main and most important entity type is the pipeline, which represents a stream processing pipeline. However, other entities, such as Connections, Streams, References and Targets are required to create and/or make effective use of the pipeline.

# Connection

A *connection* is a very basic artifact and the first entity that you need to create in the Catalog. It is a collection of metadata (such as URLs, credential and the like) required to connect to an external system. A connection is the basis for creation of sources (Streams, References or Geo Fences) and Targets.

It is possible to reuse the same connection to create multiple sources and/or targets. In other words, it can is reused to access different resources in the same system: for example different Kafka topics in the same Kafka cluster, or different database tables in the same Oracle database.

In this release, the only supported connection types are Kafka and Database.

## Kafka Connection

A Kafka connection has just a single parameter, the Zookeeper server URL above all the standard properties (name, description, tags) of catalog objects.

The Zookeper URL is of the *format host:port*. If the port is not provided by the user, the system will assume the default Zookeeper port, i.e. 2181. Authentication to Kafka is not supported in this release.

## Database Connection

Stream Analytics supports connecting to an Oracle database.

To connect to an Oracle database, you must provide the following parameters:

- Service name/SID
- hostname
- port
- username
- password

# Sources

There are two kinds of sources in Stream Analytics: streams and references. Though serve as an input to a pipeline, they are quite different. A *stream* is a representation of streaming data while a *reference* is that of static data. Streaming data is flowing into the system and is to be processed, whereas static data is used to enrich streaming data by pulling the static data from a static data source.

The initial or primary source of a pipeline must always be a stream. However additional sources can be both streams and/or references.

## Stream

A *Stream* is a source of dynamic data. The data is flowing, it is not static or frozen. For example, stock prices of a particular company can be considered as a stream as the data arrives in every second or even more frequently. Another example of streaming data is the position (geographical location) of vehicles (e.g. trucks) which again can change continuously as each vehicle is moving. Each vehicle reports its own position to a central system periodically, e.g. every second, and the central system receives the position messages as a stream.

Streams can be transmitted using different network protocols, messaging systems as well as using many different message formats. In this release, the supported stream types are: Kafka and GoldenGate.

To create a Kafka stream, you must create a Kafka connection first, and then select that connection in the stream creation wizard. In addition to the connection, the user needs to specify the Kafka topic that represents the stream of data.

## Shape

A *Shape* is the format of the data. In Stream Analytics, each message (or event, in stream processing terminology) in a Stream or Target must have the same format and this format must be specified when creating the Stream or Target. You can think of the shape as the streaming analogy of the database table structure for static data. Each shape consists of a number of fields and each field has a name and a data type. In the Stream creation wizard, it is possible to assign an alias to a field, so that the field can later be referenced by this user-given alias.

Assume that the stream contains data about orders. In this case, the shape may contain the following fields: an order id of type string, a customer id of type integer, product id of type integer, a quantity of type integer and a unit price of type Number.

## Reference

A *reference* is a source of static data that provides contextual information about the event data. The metadata and contextual information of event data is stored in a reference. In this release, the only supported reference type is an Oracle Database table. A reference is basically a link/connection to a specific database table (rather than just to a database).

References are used to enrich data that arrives from a Stream. Referring back to the previous example, the order stream contains order events and each event contains a product Id and a customer Id. Assume that there are two database tables, each containing information about the products and the customers, respectively. After creating two references, one for the products table and one for the customer table, Stream Analytics can use these references to enrich the incoming streams with information from these tables, such as product name, manufacturer, customer name, address, etc.

While references take their data from a database table, a caching mechanism can be applied. By turning on caching (a configuration option of the reference), it is possible to add a caching layer in between the pipeline and the database table. This improves the performance of accessing static data, at the price of higher memory consumption by the pipeline. Once the data is loaded into cache, the reference fetches data from the cache only. Any update on the reference table does not take effect.

## Geo Fence

A *geo fence* is a virtual boundary in a real world geographical area. This virtual boundary can be used to find object position with respect to the geo fence.

For example, the object position can be:

- Near to geo fence
- Exit geo fence
- Based on Stay Duration in geo fence
- Enters geo fence
- Present inside geo fence

## Target

A *target* represents an external system where the results of the stream processing is being directed to. Just like streams, targets are the links to the outside world. Streams are the input to a pipeline, whereas targets are the output. While a pipeline can consume and process multiple streams, as of this release, a pipeline can have maximum one target.

It can have no target, but that configuration does not really make sense, as the purpose of creating a pipeline is to process streaming data and direct the output to an external system, i.e a target.

## Pipeline

A *pipeline* defines the pipeline logic and is a sequence of data processing stages. A stage can be one of the following types – Query, Pattern, Rule, Query Group.

A pipeline always starts with a stream and can optionally end with a target. The output stream of one stage is used as an input to another stage and a pipeline can be of arbitrary length with any combination of above stage types.

You can edit/update configuration on any stage, not limited to last stage (the stage before the target) in a draft pipeline.

## Query Stage

A *query stage* is used to configure a SQL-like query on the data stream and comprises additional sources for joins, filters, summaries, group by, time windows, and so on.

For example, the query below calculates hourly total sales where transaction amount is greater than a dollar and outputs the result every 1 second.

```
Select sum (TransactionAmount) As HourlySales
From SalesStream [Range 1 Hour Slide 1 Second]
Where TransactionAmount > 1
```

Queries like above or more complex queries can all be configured in the query stage with zero coding and with no intimate knowledge of Continuous Query Language or

CQL. The CQL language is similar to SQL but with additional constructs for temporal analytics and pattern matching.

A query stage has the following sub sections:

- Filter
- Correlation
- Summary/Group By
- Range
- Evaluation Frequency

## Filter

The *filter* section in a query stage or query group stage allows events in the data stream to be filtered out.

Only events which satisfy the filter condition are passed to the downstream stage. For example, in a data stream containing `SensorId` and `Temperature`, you can filter events where `Temperature` is lower than or equal to 70 degrees by setting the filter condition to `Temperature > 70`.

## Correlation

A *correlation* is used to enrich the incoming event in the data stream with static data in a database table or with data from other streams.

For example, if the event in the data stream only includes `SensorId` and `Sensor Temperature`, the event could be enriched with data from a table to obtain `SensorMake`, `SensorLocation`, `SensorThreshold`, and many more.

Correlating an event with other sources requires the join condition to be based on a common key. In the above example, the `SensorId` from the stream cand be used to correlate with `SensorKey` in the database table. The following query illustrates the above data enrichment scenario producing sensor details for all sensors whose temperature exceeds their pre-defined threshold.

```
Select T.SensorId, T.Temperature, D.SensorName, D.SensorLocation
From TemperatureStream[Now] T, SensorDetailsTable D
Where T.SensorId = D.SensorKey And T.Temperature > D.SensorThreshold
```

Queries like above and more complex queries can be automatically generated by configuring sources and filter sections of the query stage.

## Summary

A data stream is a continuous sequence of events but we can summarize the data over any time range including an unbounded range.

For example, you can continuously compute the maximum temperature for each sensor from the beginning of time by configuring a query like the one below in a Query stage.

```
Select SesnsorId, max(Temperature)
From TemperatureStream
Group By SensorId
```

## Group By

A *group by* collects the data of all the rows with an identical column value. Group by is used in conjunction with Summaries (aggregate functions) to provide information about each group.

Here is an example configuration that generates a query for computing the average temperature of each sensor at the end of the hour and using readings from last one hour.

```
Select SesnsorId, avg(Temperature)
From TemperatureStream [Range 1 Hour Slide 1 Hour]
Group By SensorId
```

**Example**

If you add multiple group bys, the data is grouped on multiple columns. For example, you have a stream that gives you sales numbers for geographical locations. You have the following columns BEFORE group by:

```
COUNTRY     CITY       REVENUE
US            SF          500
US            NY          1000
INDIA        BOMBAY    800
INDIA        BOMBAY    1500
INDIA        BOMBAY    700
.........
```

Calculate sum of revenue (summary) by country (groupby) to get:

```
COUNTRY     SUM_OF_REVENUE
US            1500
INDIA        3000
```

Add CITY as another group by, to get your aggregations grouped by city in addition to country:

```
COUNTRY        CITY          SUM_OF_REVENUE
US               NY             1000
US               SF             500
INDIA           BOMBAY      1500
INDIA          BANGALORE   1500
```

## Range

A *range* is a window applied on the data stream. Since data stream is an unbounded sequence of events it is often necessary to apply a window when computing aggregates.

Examples of ranges include – `Last 1 Hour` of events, `Last 5 Minutes` of events, `Last 10 Events`, and many more. Applying a range retains data in memory so be cautious with use of window ranges. For example, if data is arriving at the rate of 2000 events per second and if each event is 1KB then we have 2MB of data per second. Applying a 1-hour window on this data stream consumes 2MB times 3600 or 7.2GB of memory.

The supported time units in a range are:

- now
- nanoseconds
- microseconds
- milliseconds
- seconds
- minutes
- hours
- events

## Evaluation Frequency

Evaluation Frequency or a Window Slide (commonly referred to) determines the frequency at which results are desired.

For example, the configured query below outputs total sales every 1 second but using transactions from last 1 hour.

```
Select sum (TransactionAmount) As HourlySales
From SalesStream [Range 1 Hour Slide 1 Second]
```

In other words, Evaluation Frequency determines how often you want to see the results. In the above query, if result is only desired at the end of the hour then we set the Evaluation Frequency to 1 hour.

# Rule Stage

A *rule stage* is a stage in the pipeline where you apply conditional (`IF - THEN`) logic to the events in the stream. You can check for specific conditions and assign values to fields based on the results of your checks.

You can add multiple rules to the stage and they will get applied to pipeline in the sequence they are added.

## Rule

A *rule* is a set of conditions applied to the incoming stream and a set of actions performed on the stream when conditions are true. Each event is analyzed independently of other events.

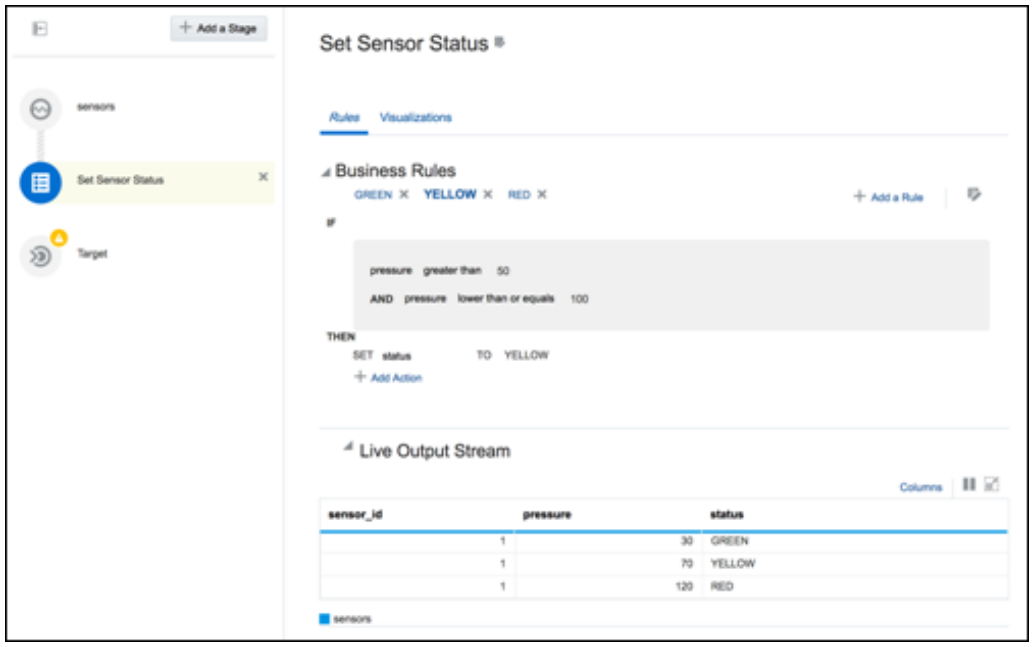
For example, assume that your stream is a stream from pressure sensors and has the following fields:

- `sensor_id`
- `pressure`
- `status`

If you want to assign a status value based on the pressure, you can define the following rules:

- if the pressure is less than or equal to 50, the status must be set to GREEN
- if the pressure is between 50 and 100, the status must be set to YELLOW
- if the pressure is greater than 100, the status must be set to RED.

To achieve this, you need to create these rules in a rule stage. The YELLOW rule for example, looks as shown below:




The rules get applied to the events sequentially and actions are triggered if the conditions are met. If you look at the data in the previous screen, the pressure value is 120 in the last row and hence the RED rule conditions resolve to *true*.

You must be careful while defining the rules. Logical loops or contradictory rules lead to the application never returning any outgoing events. For example, the following rules force the application into running forever without any outgoing events:

- Rule 1: if n > 0, set n to -1
- Rule 2: if n <=0, set n to 1

## Pattern Stage

Patterns are a stage within a pipeline. When working from a pattern, you need to specify a few key fields to discover an interesting result. You can create pattern stages within the pipeline. Patterns are not stand-alone artifacts. They need to be embedded within a pipeline.

## Query Group

A query group stage lets you do aggregations on multiple group bys and multiple windows. It is a collection of groups, where each of the group has its own window, filters that affect the data processing only within that group.

A query group has two types of stages:

- Stream
- Table

### Query Group Stage: Stream

A query group stage of the type stream is where you can apply aggregate functions with different group-bys and window ranges to your streaming data. You can have multiple query groups in one stage.

### Query Group Stage: Table

A query group stage of the type table is where you can apply aggregate functions with different group bys and window ranges to a database table data recreated in memory. Use this stage on a change data capture stream, such as GoldenGate. You can have multiple query groups in one stage.

## Live Output Table

The Live Output table is the main feedback mechanism from the pipelines that you build. The Live Output table will display events that go out of your pipeline, after your processing logic has been applied on the incoming stream or streams.

The Live Output table will be displayed for each stage of your pipeline and will include output of that particular stage. On the source stage the Live Output table will display events as they arrive in the stream. On the target stage, the Live Output stage will display events as they will flow to the target.

The Live Output table is also a powerful tool for event shape manipulation. With the Live Output table you can:

- Add new fields to the event using an extensive library of functions in the expression builder, and remove new fields
- Change the order of the event fields
- Rename event fields
- Remove existing fields from the output of the stage
- Add a timestamp field to each event

- Hide fields from view (but retain them in the output)
- Pause and restart event display in the browser (not affecting downstream stages or targets)

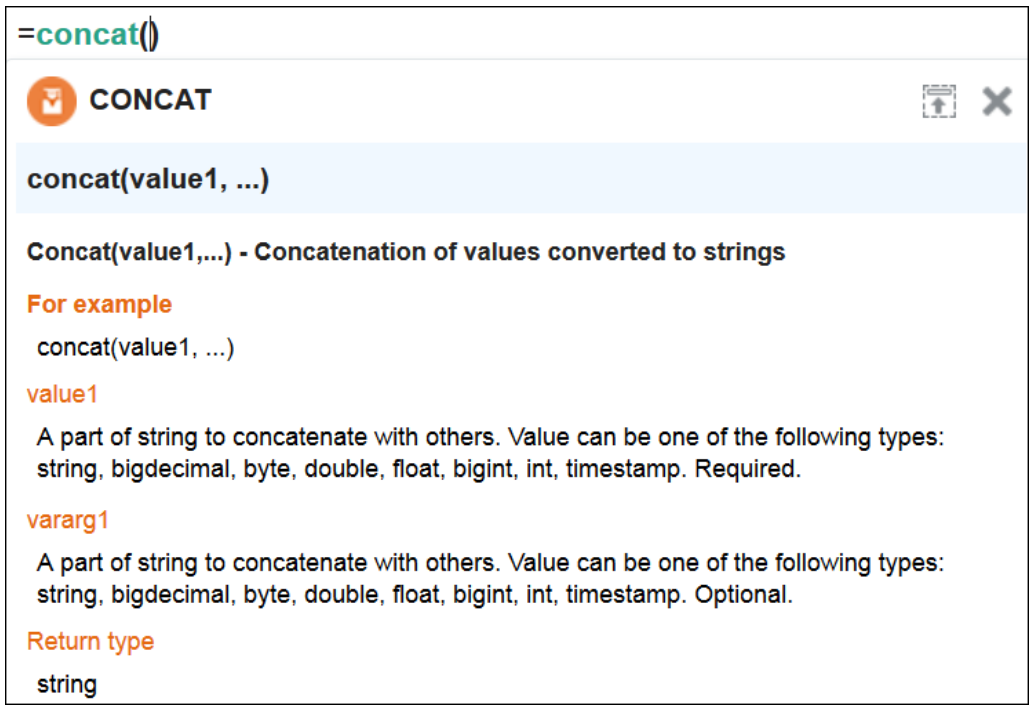
The interaction with the table should be intuitively clear to anyone who has worked with popular spreadsheet pipelines.

### Expression Builder

The expression builder provides functionality to add new fields to your output based on existing fields. You can use a rich library of functions to manipulate your event data. A simple example is string concatenation; you can construct a full name from first and last names:


> **Note:**
>
> The event shape manipulation functionality is available on the table in the query stage.

The expression builder has syntax highlighting and code completion. You can also see the function signature, input parameters and the return value in the Expression Editor.



# Visualizations

Visualization is mapping of the data (information) to a graphical or tabular format which can be used to answer a specific analytical question.

It translates data, its properties and relationships into an easy to interpretable visual object consisting of points, lines, shapes and colors. It effectively represents the results of the meaningful multi-dimensional questions. It also enables to discover the influential patterns out of the represented data (information) using the visual analysis.

### Visualizations

Visualizations are divided into two categories:

- Axis based

  Axis based visualizations display series and groups of data. Series and groups are analogous to the rows and columns of a grid of data. Typically, the rows in the grid appear as a series in visualization, and the columns in the grid appear as groups.

  Axis based visualizations enables users to visualize the data along two graph axis x and y like sum of sales over regions or sum of sales over time period. X axis values can be categorical in nature like regions or can be based on time series values whereas Y axis represents the measured value like sum(sales). These charts are useful for visualizing trends in a set of values over time and comparing these values across series.

- Spatial

  Spatial visualizations are used when geography is especially important in analyzing an event. It represents business data superimposed on a single geo fence.

### Types of Visualizations

Visualizations can be further classified into the following categories:

- Bar

  Bar visualization is one of the widely used visualization types which represents data as a series of vertical bars. It is best suited for comparison of the values represented along y axis where different categories are spread across x axis. In a Bar visualization vertical columns represent metrics (measured values). The horizontal axis displays multiple or non-consecutive categories.

  In Horizontal Bar, the axis positions are switched. The vertical axis displays multiple or non-consecutive categories. The horizontal columns represents metrics (measured values). It is preferable when the category names are long text values and requires more space in order to be displayed.

- Line

  Line visualization represents data as a line, as a series of data points, or as data points that are connected by a line. Line visualization require data for at least two points for each member in a group. The X-axis is a single consecutive dimension, such as a date-time field, and the data lines are likely to cross. X axis can also have non date-time categories. Y axis represents the metrics (measured value). It is preferred to use line visualization when data set is continuous in nature. It is best suited for trend-based plotting of data over a period of time. In Line

visualization, emphasis is on the continuation or the flow of the values (a trend) but individual value comparison can also be carried out. Multiple series can also be compared with the line visualizations.

It can have a horizontal orientation where axis are switched i.e. y axis holds categories whereas x axis shows metrics.

- Area

  Area visualization represents data as a filled-in area. Area visualization requires at least two groups of data along an axis. The X-axis is a single consecutive dimension, such as a date-time field, and the data lines are unlikely to cross. Y axis represents the metrics (measured value). X axis can also have non date-time categories. This visualization is mainly suitable for presenting accumulative value changes over time.

  It can have a horizontal orientation where axis are switched i.e. y axis holds categories whereas x axis shows metrics.

- Stacked Bar

  A Stacked visualization displays sets of values stacked in a single segmented column instead of side-by-side in separate columns. It is used to show a composition. Bars for each set of data are appended to previous sets of data. The size of the stack represents a cumulative data total.

- Spatial

  Geo Spatial visualization allows displaying location of an object on a geo fence and takes user to the area where events are occurring. User can configure visualization to specify latitude, longitude, identifier etc. Customization of visualization by specifying different pins like arrows with different colors based on certain condition is also allowed.

## Draft and Published Pipelines

Stream Analytics supports two lifecycle states, namely draft and published.

**Draft Pipelines**

Pipelines in the draft state possess the following characteristics:

- Are visible only to the owner
- Can be edited
- Work only when the pipeline editor is open. When you exit catalog or close your browser, the draft pipeline will be removed from the Spark cluster.
- Do not send events to a downstream target even if a target is configured

A newly created pipeline is in draft state. This is where you can explore your streams and implement the business logic. You do not have to do the implementation all at once; the pipeline will not run between your editing sessions.

**Published pipelines**

Pipelines in the published state possess the following characteristics:

- Are visible to any user
- Cannot be edited

- Will continue to run in the Spark cluster even after you exit the pipeline
- Send events to a downstream target

After you are done with the implementation and satisfied, you can add a target and publish your pipeline. The published pipeline runs continually on the Spark Cluster.

If you want to edit a published pipeline, you must unpublish it first.

## Oracle GoldenGate Integration

Oracle GoldenGate is a comprehensive software package for real-time data integration and replication in heterogeneous IT environments. The product set enables high availability solutions, real-time data integration, transactional change data capture, data replication, transformations, and verification between operational and analytical enterprise systems.

### Use Case

When to use Oracle GoldenGate integration in Stream Analytics? You use this integration when you want to analyze your transaction data without stressing your production database and without waiting for offline reports to be built. Stream Analytics listens to the transaction event stream pushed by Oracle GoldenGate to Kafka and provide you with operational data in real time when transactions occur.

There exists an `Orders` table in a database. The table is continually affected by transactions (insert/update/delete) and Oracle GoldenGate is replicating the transactions to Kafka. The schema for the `Orders` table is `{OrderId, OrderStatus, ProductSKU, QuantitySold, Revenue}`.

The stream from Oracle GoldenGate includes two sets of the original table columns, one to record the state before transaction (for update and delete transactions), and another to record the state after transaction (for insert and update transactions). It also includes the `op_type` and `op_ts` columns. The `op_type` column carries transactional semantics (`I` (Insert) or `U` (update) or `D` (Delete)). The `op_ts` column is the transaction timestamp.

An order can be canceled either by deleting the row or by updating the order status to `Canceled`. The default order status is `Booked`. Orders can also be revised up or down by changing quantity and therefore the revenue. Revenue goes down when quantity is lowered and goes up when quantity is increased.

### Solution

A solution is as follows:

1. Create a GoldenGate stream pointing to the Kafka topic GoldenGate replicates the transactions to.
2. Create a pipeline using the GoldenGate stream and add a Query Group Stage of the type Table. In this stage, use the transactional semantics in the `op_ts` column to rebuild the `Orders` table in memory, so that you can see a snapshot of the latest transactions in your table for as long as you specify (remember that the longer the time window, the more the memory consumed).

You can then run any filters, group bys and aggregations on this snapshot. For example:

- Total Revenue by for the past 24 hours
- Total Revenue by `ProductSKU` for the past 24 hours
- Average `QuantitySold` by `ProductSKU` for the past one hour.

> **Tip:**
>
> To enhance the data, before adding the Query Group Stage, you can add a regular Query Stage where you can join the transaction event stream with a reference database table to obtain more dimensions for your reports, such as `Product Category` or `Vendor`.

# Access Stream Analytics

Stream Analytics is a feature within Oracle Integration.

After you provision Stream Analytics, you can access it from Oracle Integration.

- If you have already registered Stream Analytics, click **Streams** in the left pane to launch and navigate to Stream Analytics.
- If you are accessing Stream Analytics for the first time, click **Register** and provide the deployment details to register an instance of Stream Analytics. Click **Streams** in the left pane to launch and navigate to Stream Analytics.


# About Stream Analytics Roles and Users

Stream Analytics delivers multiple experiences targeting the specific skillsets of every persona associated with complex business process lifecycles.

The roles and users are:

- *Developer* — has full access to the design time user interface with privileges to create, edit, and delete pipelines. The privileges do not include the Publish privilege.
- *Viewer* — has full access to the design time user interface with privileges to view pipelines and published dashboards. This role has only read access and no Publish privilege.

- *Administrator* — has administration or super user access. This is the only role allowed to modify pipeline role permission grants and memberships.
- *Architect* — has full access to the design time user interface with the privileges to create, edit, and delete all artifacts: pipelines, streams, maps, connections, references, targets, and patterns. The privileges do not include the Publish privilege.
- *Operator* — has the privileges to deploy and undeploy pipelines and to monitor the infrastructure such as the deployed pipelines, Spark Cluster, WebLogic Servers, etc.

> **Note:**
>
> A user with this role permission can publish/unpublish the pipeline only if the user also has the *Developer* role.

For additional information about the privileges, see Privileges Available to Roles in Stream Analytics.

# 2
# Administer Stream Analytics

Administering Stream Analytics is essential to get the required results.

**Topics:**



## Typical Workflow for Administering Stream Analytics

The typical workflow lists the artifacts required to create a pipeline in Stream Analytics.

The prerequisites for a pipeline are:

- A reference is used to create a stream.
- A target is required to create a connection.
- A connection is required to create a stream.
- A stream is required to create a pipeline.

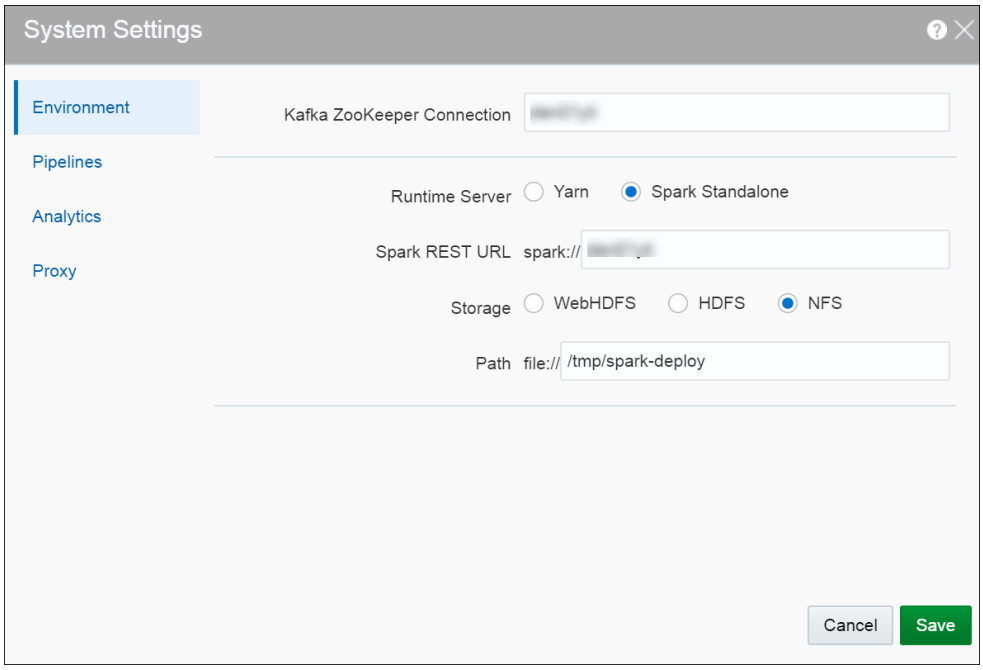
## Configure Stream Analytics System Settings

Only users with the *Administrator* role can set the console settings in Stream Analytics.

To set/update console settings:

1. Click the user name in the top right corner of the screen.
2. Click **System Settings**.

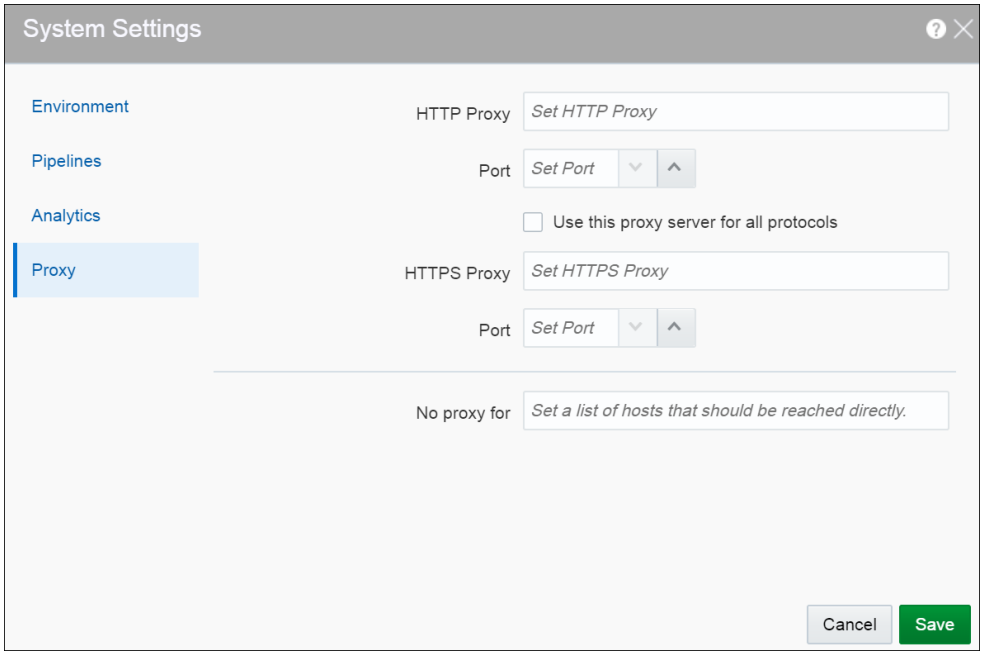
   The System Settings page opens.
3. Click **Environment**.
4. Specify the server names and URLs where the Kafka Zookeeper, Yarn Resource Manager, and Spark Standalone are deployed and running.




- **Kafka Zookeeper Connection** — the URL where the Zookeeper server or servers are configured, separated by comma. This value is required to push events to the stream.
- **Runtime Server** — the runtime server you want your Stream Analytics instance to run on
- **YARN Resource Manager URL** — the URL where the YARN Resource Manager is configured, if the runtime server is Yarn
- **Spark REST URL** — the URL where the Spark REST Proxy is configured, if the runtime server is Spark
- **Storage** — the type of storage for all your artifacts within Stream Analytics
- **Path** — the path where the storage exists
- **HA Namenodes** — the namenodes of HA cluster
- **Hadoop Authentication** — the type of Hadoop authentication you would like

5. Click **Pipelines**. Specify the various settings for the pipeline within Stream Analytics.
   - **Batch Duration** — the default duration of the batch for each pipeline
   - **Executor Count** — the default number of executors per pipeline
   - **Cores per Executor** — the default number of cores. A minimum value of 2 is required.
   - **Executor Memory** — the default allocated memory for each executor instance in megabytes
   - **Cores per Driver** — the default number of cores
   - **Driver Memory** — the default allocated memory per driver instance in megabytes
   - **High Availability** — toggle the default HA value as on/off for each pipeline

6. Click **Analytics**. Enable **Analytics** and specify the **Druid Zookeeper Connection** details. This is required to work with cubes.




7. Click **Proxy**. If you set proper proxy, the back-end system will use these settings to test the REST target.
8. Click **Save**.

# Configure User Preferences

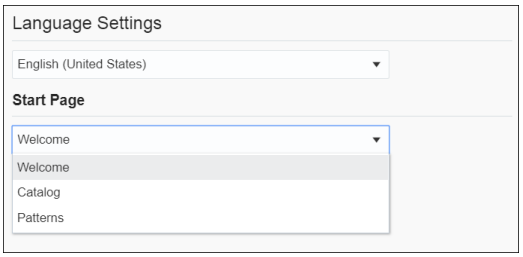
Based on the preferences that users set in this page, the characteristics of Stream Analytics vary.

To set/update user preferences:

1. Click the user name in the top right corner of the screen.
2. Click **Preferences**. The Preferences page opens.

### General

Provides a set of general preferences that you can view and set according to your requirements.

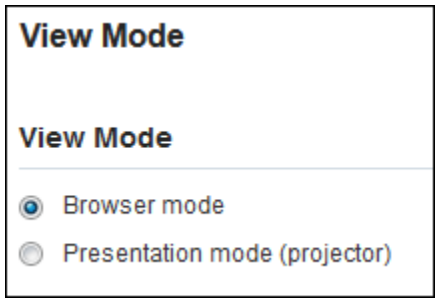



### Start Page

Select if you want the Home page, the **Catalog** page, or the **Patterns** page to appear as the **Start** Page.

### View Mode

Provides a set of view mode preferences that you can view and set according to your requirements.




### Browser mode

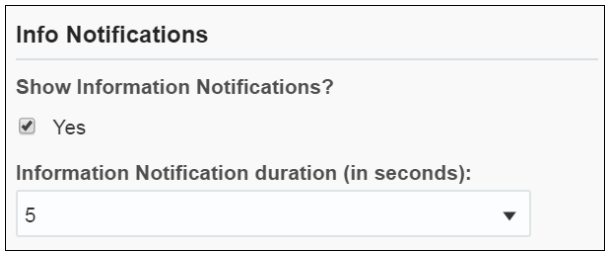
Select this option to view the pipeline in a browser mode, just like any other web pipeline appears in a browser.

### Presentation mode (projector)

Select this option to view the pipeline in a presentation mode, as if the pipeline is being viewed on a projector as a presentation. When you select the **Presentation mode**, a different color skin is applied to the pipeline.

### Notifications

Provides a set of notifications preferences that you can view and set according to your requirements.




### Show Information Notifications

Select this option if you want the information notifications to appear in the pipeline. This option is selected by default.

### Information Notification duration (in seconds)

Choose the number of seconds for which the notifications appear. The default value is 5.

### Catalog

Provides a set of catalog preferences that you can view and set according to your requirements.


**Default Sorting Column**

Select the column by which you want the columns to be sorted. This value will be used as the default for all columns until you change the value again.

**Default Sorting Order**

Select the order by which you want the columns to be sorted. This value will be used as the default value for all columns until you change the value again.

**Default Page Size**

Select the value to be used as the default page size. Based on the value selected, the number of records that appear on a page vary. This value will be used as the default for all pages until you change the value again.

**Pipeline**

Provides a set of pipeline preferences that you can view and set according to your requirements.


Select **Yes** if you want to display the User Assistance text for the pipelines in the Pipeline Editor.

**Live Output Stream**

Provides a set of pipeline live output stream preferences that you can view and set according to your requirements.

Select a value that you want to be applied as the default table size for the data in Live Output Stream of a pipeline.

**Timestamp**

Provides a set of pipeline timestamp preferences that you can view and set according to your requirements.


Timestamp function
systimestamp
Timestamp format
Use Timestamp format?
yyyy-MM-dd HH:mm:ss

3

# Work with Stream Analytics Artifacts

Stream Analytics has various artifacts like connections, references, streams, targets, and more. Artifacts are important resources that you can use to create pipelines.

**Topics:**



## About the Catalog

The Catalog page is the location where resources including pipelines, streams, references, maps, connections, and targets are listed. This is the go-to place for you to perform any tasks in Stream Analytics.

You can mark a resource as a favorite in the Catalog by clicking on the Star icon. Click the icon again to remove it from your favorites. You can also delete a resource or view its topology using the menu icon to the right of the favorite icon.

The tags applied to items in the Catalog are also listed on the screen below the left navigation pane. You can click any of these tags to display only the items with that tag in the Catalog. The tag appears at the top of the screen. Click **Clear All** at the top of the screen to clear the Catalog and display all the items.

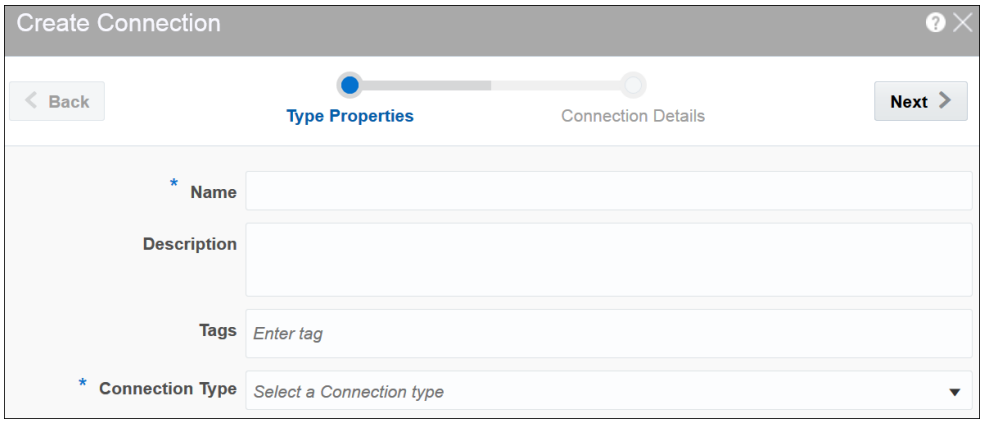
You can include or exclude pipelines, streams, references, maps, connections, and targets using the **View All** link in the left panel under **Show Me**. When you click **View All**, a check mark appears beside it and all the components are displayed in the Catalog.

When you want to display or view only a few or selective items in the Catalog, deselect **View All** and select the individual components. Only the selected components will appear in the Catalog.

# Create a Connection

To create a connection:

1. Click **Catalog** in the left pane.
2. From the **Create New Item** menu, select **Connection**.
3. Provide details for the following fields on the **Type Properties** page and click **Next**:
   - **Name** — name of the connection
   - **Description** — description of the connection
   - **Tags** — tags you want to use for the connection
   - **Connection Type** — type of connection: Database or Kafka




4. Enter **Connection Details** on the next screen and click **Save**.

   When the connection type is **Kafka**, provide Zookeeper URLs.

   When the connection type is **Database**:

   - **Connect using** — select the way you want to identify the database; `SID` or `Service name`
   - **Service name/SID** — the details of the service name or SID
   - **Host name** — the host name on which the database is running
   - **Port** — the port on which the database is running. Usually it is `1521`
   - **Username** — the user name with which you connect to the database
   - **Password** — the password you use to login to the database

A connection with the specified details is created.

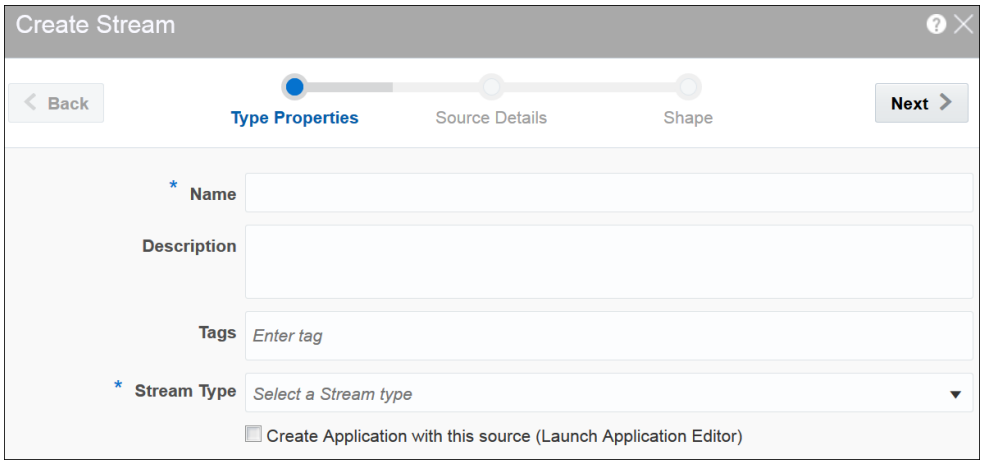
# Create a Stream

A stream is a source of events with a given content (shape).

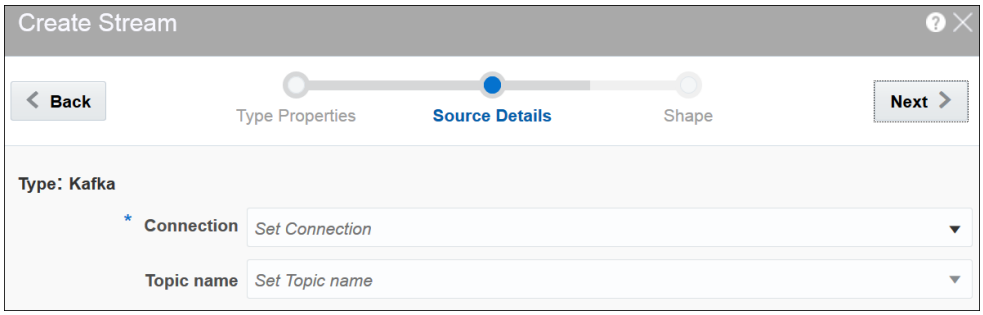
To create a stream:

1. Navigate to **Catalog**.

2. Select **Stream** in the **Create New Item** menu.
3. Provide details for the following fields on the **Type Properties** page and click **Next**:
   - **Name** — name of the stream
   - **Description** — description of the stream
   - **Tags** — tags you want to use for the stream
   - **Stream Type** — select suitable stream type. Supported types are Kafka and GoldenGate




4. Provide details for the following fields on the **Source Details** page and click **Next**:
   - **Connection** — the connection for the stream
   - **Topic Name** — the topic name that receives events you want to analyze

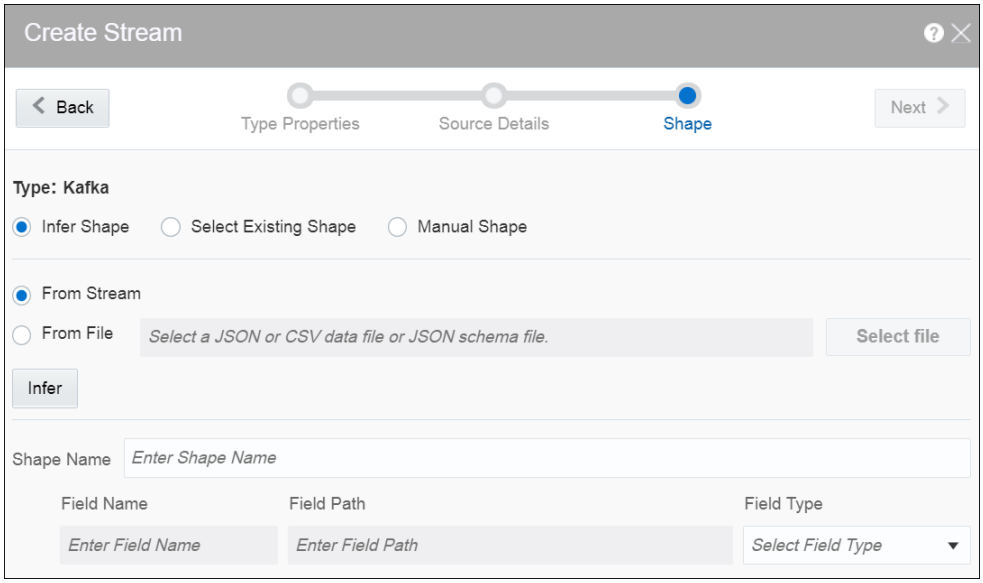



5. Select one of the mechanisms to define the shape on the **Shape** page:
   - **Infer Shape**
   - **Select Existing Shape**
   - **Manual Shape**

   **Infer Shape** detects the shape automatically from the input data stream. You can infer the shape from Kafka or from JSON schema/message in a file. You can also save the auto detected shape and use it later.

   **Select Existing Shape** lets you choose one of the existing shapes from the drop-down list.

**Manual Shape** populates the existing fields and also allows you to add or remove columns from the shape. You can also update the datatype of the fields.



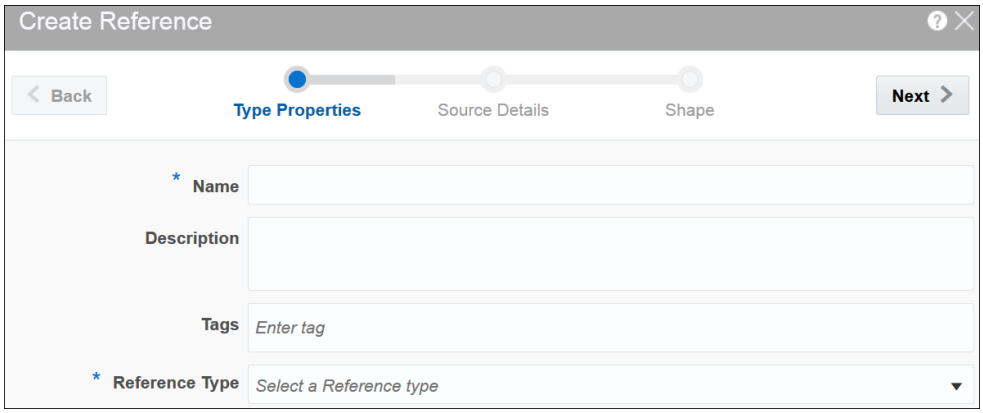

A stream is created with specified details.

# Create a Reference

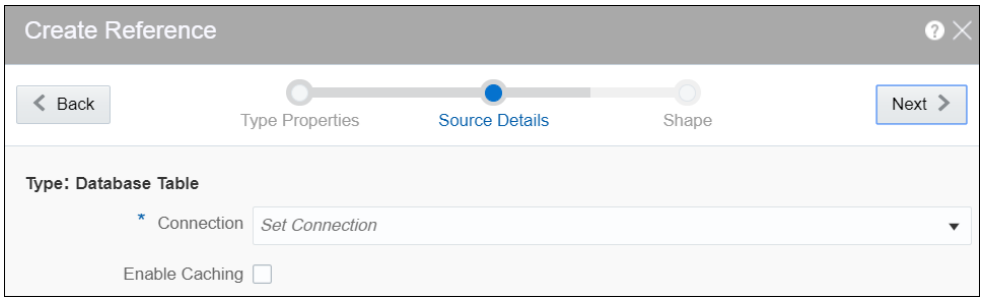
A reference defines a read-only source of reference data to enrich a stream. A stream containing a customer name could use a reference containing customer data to add the customer's address to the stream by doing a lookup using the customer name. A reference currently can only refer to database tables. A reference requires a database connection.

To create a reference:

1. Navigate to **Catalog**.
2. Select **Reference** in the **Create New Item** menu.
3. Provide details for the following fields on the **Type Properties** page and click **Next**:
   - **Name** — name of the reference
   - **Description** — description of the reference
   - **Tags** — tags you want to use for the reference
   - **Reference Type** — the reference type of the reference



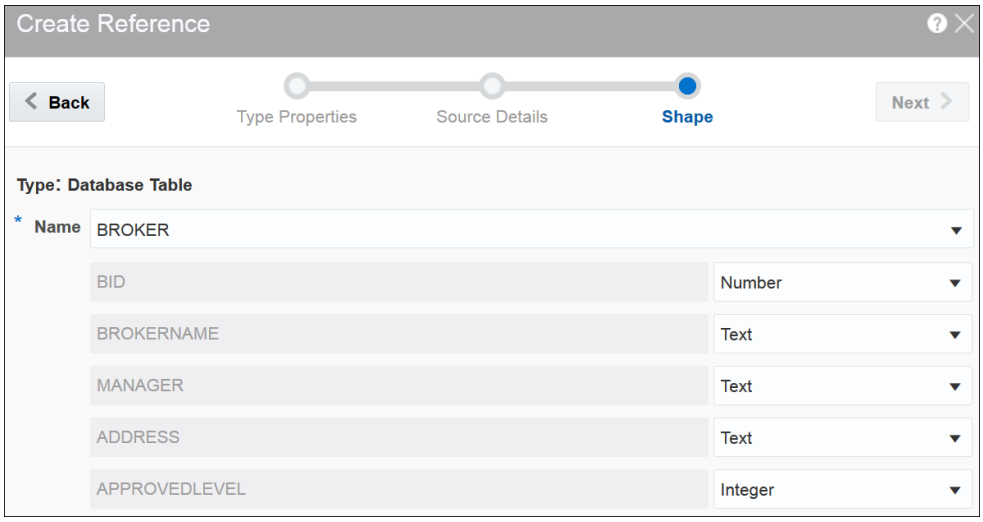

4. Provide details for the following fields on the **Source Details** page and click **Next**:
   - **Connection** — the connection for the stream




   - **Enable Caching** — select this option to enable caching for better performance at the cost of higher memory usage of the Spark applications. Caching is supported only for single equality join condition. When you enable caching, any update to the reference table does not take effect as the data is fetched from the cache.

5. Provide details for the following fields on the **Shape** page and click **Save**:
   - **Name** — name of the database table

> **NOT_SUPPORTED:**
>
> Ensure that you do not use any of the CQL reserved words as the column names. If you use the reserved keywords, you cannot deploy the pipeline.




When the datatype of the table data is not supported, the table columns do not have auto generated datatype. Only the following datatypes are supported:

- `numeric`
- `interval day to second`
- `text`
- `timestamp` (without timezone)
- `date time` (without timezone)

A reference is created with the specified details.

# Create a Dashboard

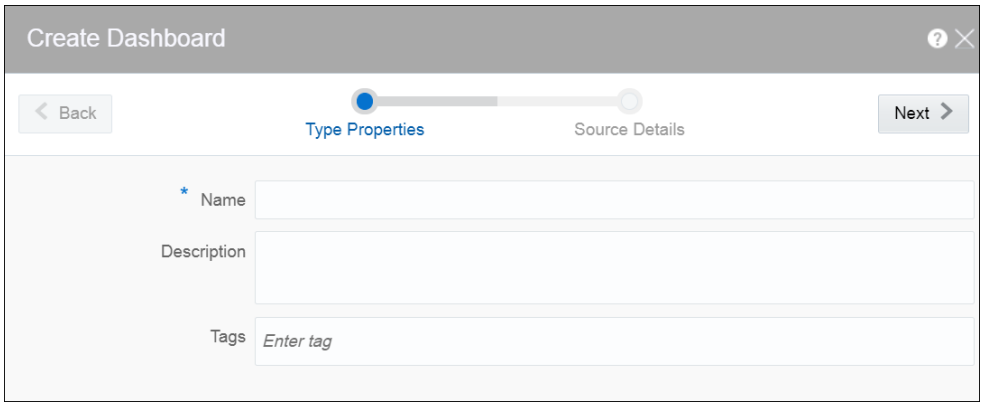
A dashboard is a visualization tool that helps you look at and analyze the data related to a pipeline based on various metrics like slices.

A dashboard is an analytics feature. You can create dashboards in Stream Analytics to have a quick view at the metrics.

To create a dashboard:

1. Go to the **Catalog**.
2. Select **Dashboard** in the **Create New Item** menu.

   The Create Dashboard screen appears.




3. Provide suitable details for the following fields:
   - **Name** — enter a name for the dashboard. this is a mandatory field.
   - **Description** — enter a suitable description for the dashboard. This is an optional field.
   - **Tags** — enter or select logical tags to easily identify the dashboard in the catalog. This is an optional field.
4. Click **Next**.
5. Enter a custom stylesheet for the dashboard. This is an optional step.
6. Click **Save**.

   You can see the dashboard in the Catalog.

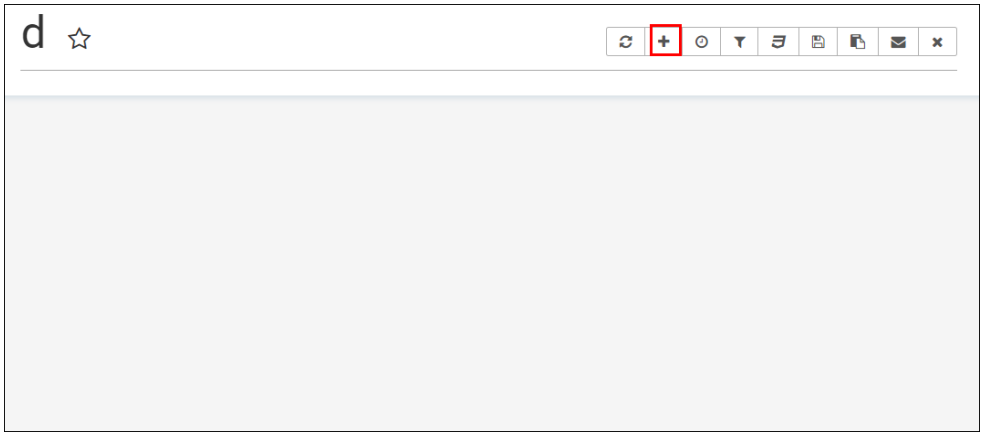
After you have created the dashboard, it is just an empty dashboard. You need to start adding details to the dashboard.

***Editing a Dashboard***

To edit a dashboard:

1. Click the required dashboard in the catalog.

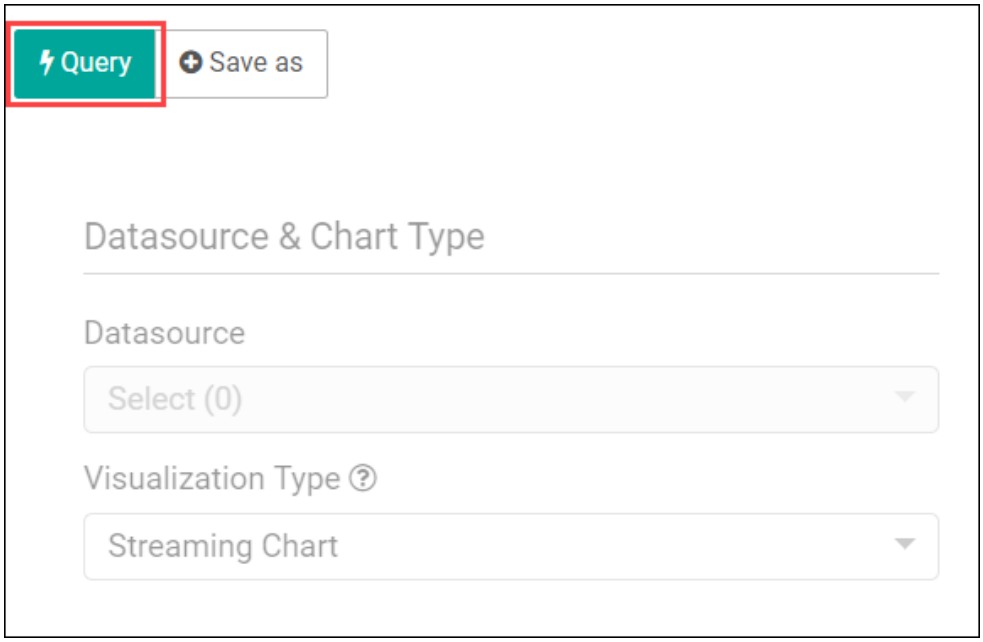
   The dashboard opens in the dashboard editor.




2. Click the **Add a new slice to the dashboard** icon to see a list of existing slices. Go through the list, select one or more slices and add them to the dashboard.
3. Click the **Specify refresh interval** icon to select the refresh frequency for the dashboard.

This just a client side setting and is not persisted with the Superset Version `0.17.0`.

4. Click the **Apply CSS to the dashboard** icon to select a CSS. You can also edit the CSS in the live editor.
5. Click the **Save** icon to save the changes you have made to the dashboard.
6. Within the added slice, click the **Explore chart** icon to open the chart editor of the slice.




You can see the metadata of the slice.

7. Click **Save as** to make the following changes to the dashboard:
   a. Overwrite the current slice with a different name
   b. Add the slice to an existing dashboard
   c. Add the slice to a new dashboard

# Create a Cube

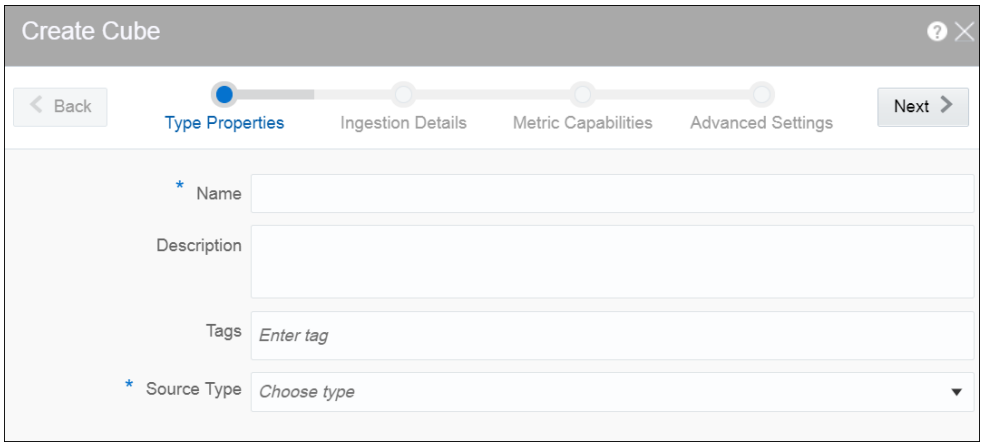
A cube is a data structure that helps in quickly analyzing the data related to a business problem on multiple dimensions.

The cube feature works only when you have enabled Analytics. Verify this in System Settings.

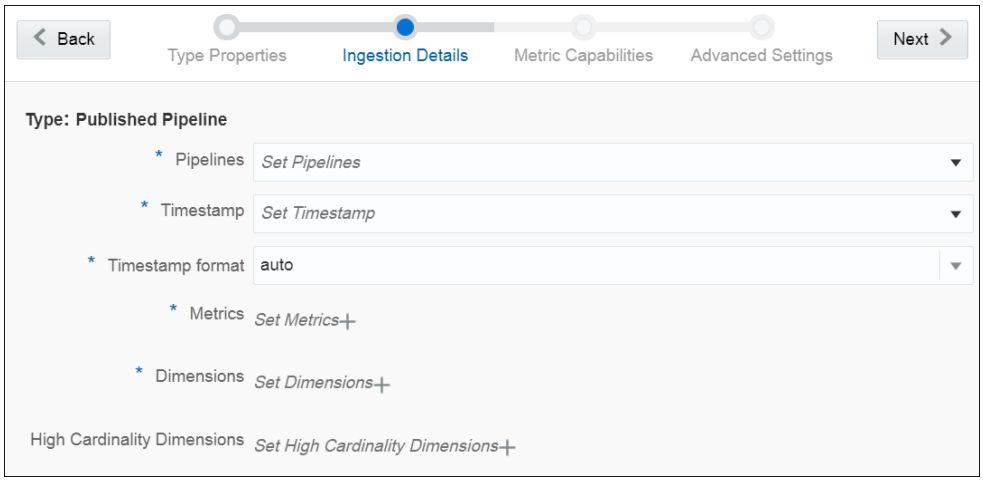
To create a cube:

1. Go to the **Catalog**.
2. From the **Create New Item** menu, select **Cube**.
3. On the Create Cube — Type Properties screen, provide suitable details for the following fields:
   - **Name** — enter a name for the cube. This is a mandatory field.

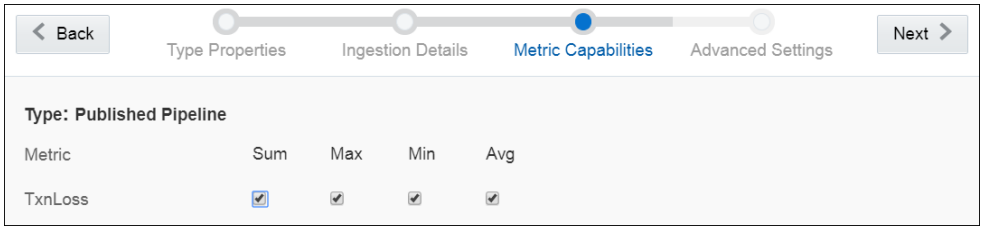
- **Description** — enter a suitable description for the cube. This is an optional field.
- **Tags** — enter or select logical tags for the cube. This is an optional field.
- **Source Type** — select the source type from the drop-down list. Currently, Published Pipeline is the only supported type. This is a mandatory field.



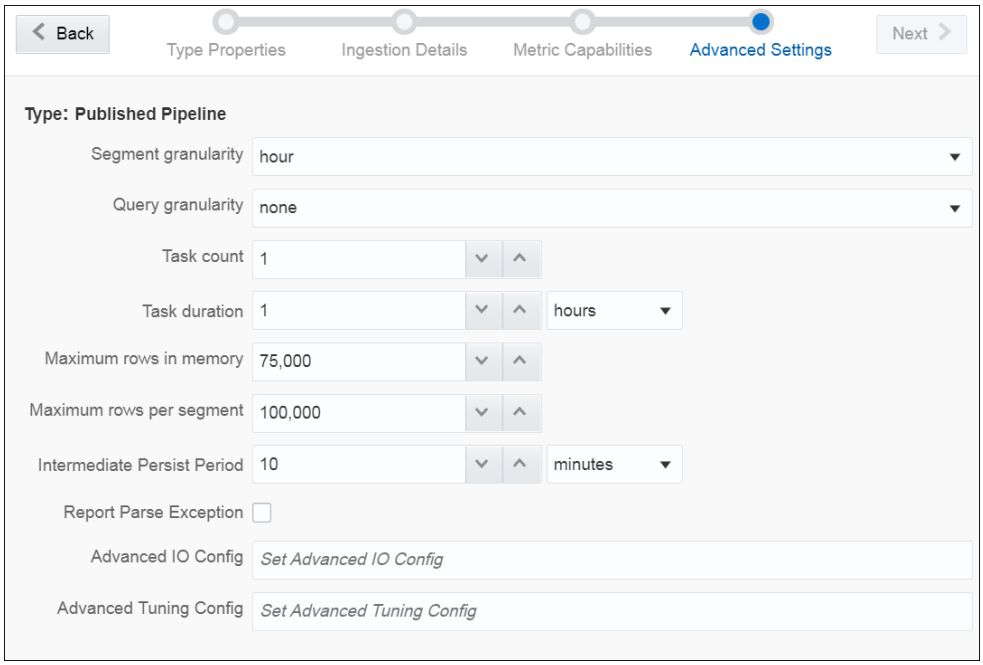

4. Click **Next** and provide suitable details for the following fields on the Ingestion Detailsscreen:
   - **Pipelines** — select a pipeline to be used as the base for the cube. This is a mandatory field.
   - **Timestamp** — select a column from the pipeline to be used as the timestamp. This is a mandatory field.
   - **Timestamp format** — select or set a suitable format for the timestamp using Joda time format. This is a mandatory field. *auto* is the default value.
   - **Metrics** — select metrics for creating measures
   - **Dimensions** — select dimensions for group by
   - **High Cardinality Dimensions** — high cardinality dimensions such as unique IDs. Hyperlog approximation will be used.




5. Click **Next** and select the required values for the **Metric** on the Metric Capabilities screen.




6. Click **Next** and make any changes, if required, on the Advanced Settings screen.
   - **Segment granularity** — select the granularity with which you want to create segments
   - **Query granularity** — select the minimum granularity to be able to query results and the granularity of the data inside the segment
   - **Task count** — select the maximum number of reading tasks in a replica set. This means that the maximum number of reading tasks is `taskCount*replicas` and the total number of tasks (reading + publishing) is higher than this. The number of reading tasks is less than `taskCount if taskCount > {numKafkaPartitions}`.
   - **Task duration** — select the length of time before tasks stop reading and begin publishing their segment. The segments are only pushed to deep storage and loadable by historical nodes when the indexing task completes.
   - **Maximum rows in memory** — enter a number greater than or equal to 0. This number indicates the number of rows to aggregate before persisting. This number is the post-aggregation rows, so it is not equivalent to the number of input events, but the number of aggregated rows that those events result in. This is used to manage the required JVM heap size. Maximum heap memory usage for indexing scales with `maxRowsInMemory*(2 + maxPendingPersists)`.
   - **Maximum rows per segment** — enter a number greater than or equal to 0. This is the number of rows to aggregate into a segment; this number is post-aggregation rows.
   - **Immediate Persist Period** — select the period that determines the rate at which intermediate persists occur. This allows the data cube is ready for query earlier before the indexing task finishes.
   - **Report Parse Exception** — select this option to throw exceptions encountered during parsing and halt ingestion.
   - **Advanced IO Config** — specify name-value pair in a CSV format. Available configurations are `replicas`, `startDelay`, `period`, `useEarliestOffset`, `completionTimeout`, and `lateMessageRejectionPeriod`.
   - **Advanced Tuning Config** — specify name-value pair in CSV format. Available configurations are `maxPendingPersists`, `handoffConditionTimeout`, `resetOffsetAutomatically`, `workerThreads`, `chatThreads`, `httpTimeout`, and `shutdownTimeout`.




7. Click **Save** to save the changes you have made.

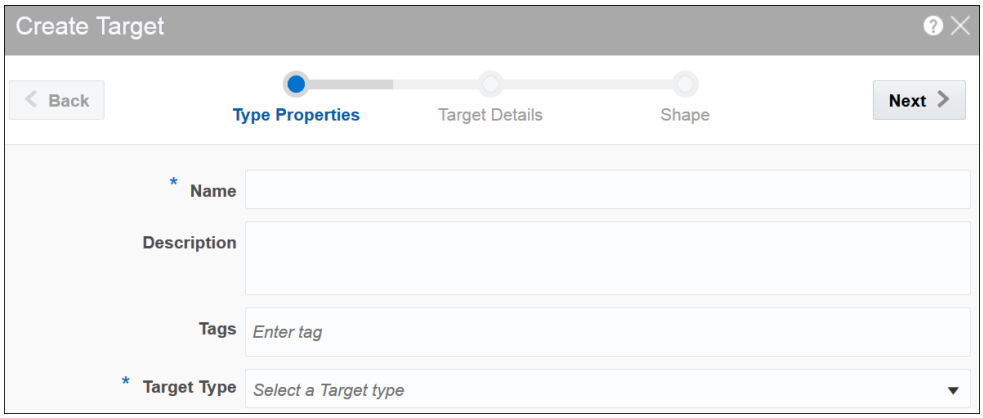
You can see the cube you have created in the catalog.

# Create a Target

A target defines a destination for output data coming from a pipeline.

To create a target:

1. Navigate to **Catalog**.
2. Select **Target** in the **Create New Item** menu.
3. Provide details for the following fields on the **Type Properties** page and click **Save** and **Next**:
   - **Name** — name of the target
   - **Description** — description of the target
   - **Tags** — tags you want to use for the target
   - **Target Type** — the transport type of the target. Supported types are Kafka and REST.




4. Provide details for the following fields on the **Target Details** page and click **Next**:

   When the target type is Kafka:

   - **Connection** — the connection for the target
   - **Topic Name** — the Kafka topic to be used in the target


   When the target type is REST:

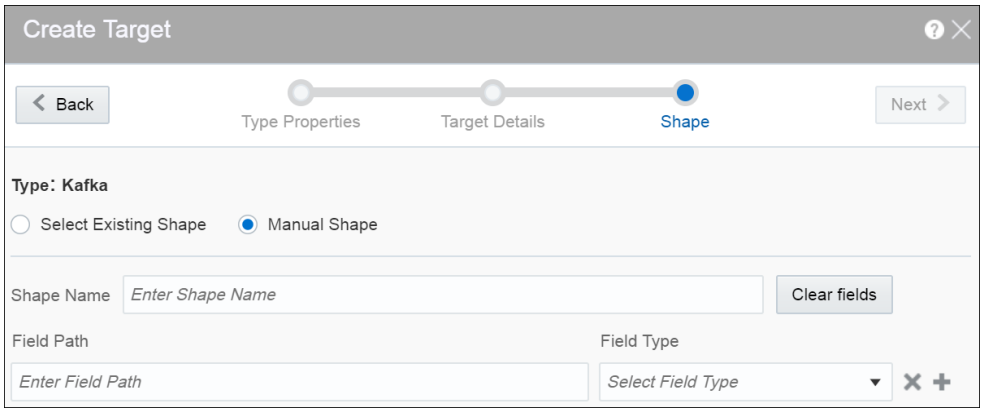
   - **URL** — enter the REST service URL. This is a mandatory field.
   - **Custom HTTP headers** — set the custom headers for HTTP. This is an optional field.
   - **Batch processing** — select this option to send events in batches and not one by one. Enable this option for high throughput pipelines. This is an optional field.

   Click **Test connection** to check if the connection has been established successfully.

   Testing REST targets is a heuristic process. It uses proxy settings. The testing process uses GET request to ping the given URL and returns success if the server returns `OK (status code 200)`. The return content is of the type of `application/json`.

5. Select one of the mechanisms to define the shape on the **Shape** page and click **Save**:

   - **Select Existing Shape** lets you choose one of the existing shapes from the drop-down list.
   - **Manual Shape** populates the existing fields and also allows you to add or remove columns from the shape. You can also update the datatype of the fields.




A target is created with specified details.

### Creating Target from Pipeline Editor

Alternatively, you can also create a target from the pipeline editor. When you click Create in the target stage, you are navigated to the Create Target dialog box. Provide all the required details and complete the target creation process. When you create a target from the pipeline editor, the shape gets pre-populated with the shape from the last stage.

# Create a Geo Fence

Geo fences are further classified into two categories: manual geo fence and database-based geo fence.

### Create a Manual Geo Fence

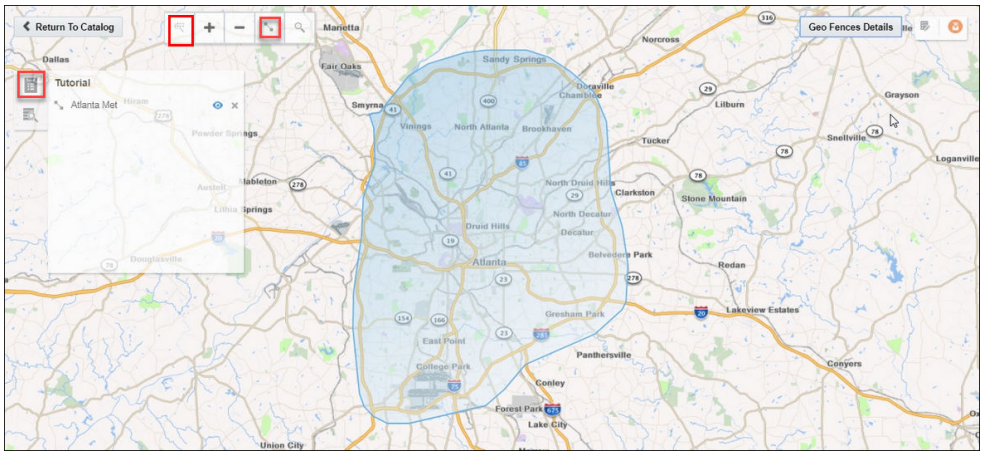
To create a manual geo fence:

1. Navigate to the **Catalog** page.
2. Click **Create New Item** and select **Geo Fence** from the drop-down list.

   The Create Geo Fence dialog opens.
3. Enter a suitable name for the **Geo Fence**.
4. Select **Manually Created Geo Fence** as the **Type**.
5. Click **Save**.

   The Geo Fence Editor opens. In this editor you can create the geo fence according to your requirement.
6. Within the Geo Fence Editor, **Zoom In** or **Zoom Out** to navigate to the required area using the zoom icons in the toolbar located on the top-left side of the screen.

   You can also use the **Marquee Zoom** tool to move across locations on the map.
7. Click the **Polygon Tool** and mark the area around a region to create a geo fence.




8. Enter a name and description, and click **Save** to save your changes.

### Update a Manual Geo Fence

To update a manual geo fence:

1. Navigate to the **Catalog** page.
2. Click the name of the geo fence you want to update.

   The Geo Fence Editor opens. You can edit/update the geo fence here.

### Search Within a Manual Geo Fence

You can search the geo fence based on the country and a region or address. The search field allows you search within the available list of countries. When you click the search results tile in the left center of the geo fence and select any result, you are automatically zoomed in to that specific area.

### Delete a Manual Geo Fence

To delete a manual geo fence:

1. Navigate to **Catalog** page.
2. Click **Actions**, then select **Delete Item** to delete the selected geo fence.

### Create a Database-based Geo Fence

To create a database-based geo fence:

1. Navigate to **Catalog** page.
2. Click **Create New Item** and then select **Geo Fence** from the drop-down list.

   The Create Geo Fence dialog opens.
3. Enter a suitable name for the geo fence.
4. Select **Geo Fence** from Database as the Type.
5. Click **Next** and select **Connection**.
6. Click **Next**.

   All tables that have the field type as `SDO_GEOMETERY` appear in the drop-down list.
7. Select the required table to define the shape.

8. Click **Save**.

> **Note:**
>
> You cannot edit/update database-based geo fences.

**Delete a Database-based Geo Fence**

To delete a database-based geo fence:

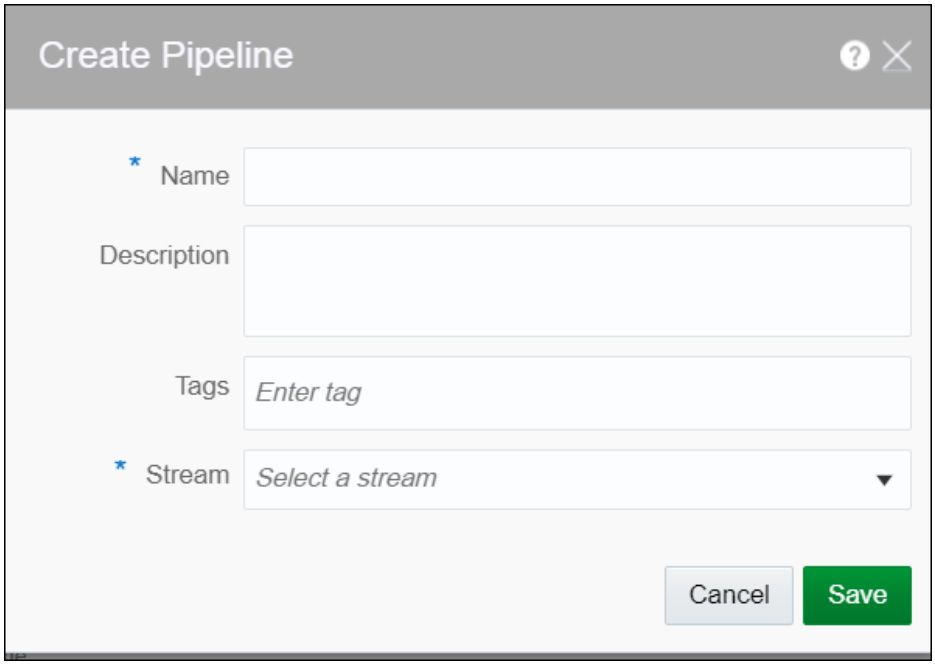
1. Navigate to **Catalog** page.
2. Click **Actions** and then select **Delete Item** to delete the selected geo fence.

# Create a Pipeline

A pipeline is a Spark application where you implement your business logic. It can have multiple stages such as a query, a pattern stage, a business rule, or a query group.

To create a pipeline:

1. Navigate to **Catalog**.
2. Select **Pipeline** in the **Create New Item** menu.
3. Provide details for the following fields and click **Save**:
   - **Name** — name of the pipeline
   - **Description** — description of the pipeline
   - **Tags** — tags you want to use for the pipeline
   - **Stream** — the stream you want to use for the pipeline




A pipeline is created with specified details.

# Configure a Pipeline

You can configure a pipeline to use various stages like query, pattern, rules, query group.

## Add a Query Stage

You can include simple or complex queries on the data stream without any coding to obtain refined results in the output.

1. Open a pipeline in the **Pipeline Editor**.
2. Click the **Add a Stage** button and select **Query**.
3. Enter a **Name** and **Description** for the Query Stage.
4. Click **Save**.

## Adding and Correlating Sources and References

You can correlate sources and references in a pipeline.

To add a correlating source or reference:

1. Open a pipeline in the **Pipeline Editor**.
2. Select the required query stage.
3. Click the **Sources** tab.
4. Click **Add a Source**.
5. Select a source (stream or reference) from the available list.
6. Click the **Window Area** in the source next to the clock icon and select appropriate values for **Range** and **Evaluation Frequency**.
7. Under **Correlation Conditions**, select **Match All** or **Match Any** as per your requirement. Then click **Add a Condition**.
8. Select the fields from the sources and the appropriate operator to correlate.

   Ensure that the fields you use on one correlation line are of compatible types. The fields that appear in the righ drop-down list depend on the field you select in the left drop-down list.
9. Repeat these steps for as many sources or references as you want to correlate.

## Adding Filters

You can add filters in a pipeline to obtain more accurate streaming data.

To add a filter:

1. Open a pipeline in the **Pipeline Editor**.
2. Select the required query stage.
3. Navigate to the **Filters** tab.
4. Click **Add a Filter**.

5. Select the required column and a suitable operator and value.

   You can also calculated fields within filters.
6. Click **Add a Condition** to add and apply a condition to the filter.
7. Click **Add a Group** to add a group to the filter.
8. Repeat these steps for as many filters, conditions, or groups as you want to add.

## Adding Summaries

To add a summary:

1. Open a pipeline in the **Pipeline Editor**.
2. Select the required query stage and click the **Summaries** tab.
3. Click **Add a Summary**.
4. Select the suitable function and the required column.
5. Repeat the above steps to add as many summaries you want.

## Adding Group Bys

To add a group by:

1. Open a pipeline in the **Pipeline Editor**.
2. Select the required query stage and click the **Summaries** tab.
3. Click **Add a Group By**.
4. Click **Add a Field** and select the column on which you want to group by.

   A group by is created on the selected column.

When you create a group by, the live output table shows the group by column alone by default. Turn ON **Retain All Columns** to display all columns in the output table.

You can add multiple group bys as well.

## Adding Visualizations

Visualizations are graphical representation of the streaming data in a pipeline. You can add visualizations on all stages in the pipeline.

To add a visualization:

1. Open a pipeline in the **Pipeline Editor**.
2. Select the required stage and click the **Visualizations** tab.
3. Click **Add a Visualization**.
4. Select a suitable visualization type from the available list.
   - Bar Chart
   - Line Chart
   - Geo Spatial
   - Area Chart

- Pie Chart
- Scatter Chart
- Bubble Chart
- Stacked Bar Chart

5. Provide all the required details to populate data in the visualization.
6. Select **Horizontal** if you want the visualization to appear with a horizontal orientation in the Pipeline Editor. This is optional and you can decide based on your usecase or requirement if you want to change the orientation.
7. Select **Save as Slice** check box if you want the visualization to be available as a slice for future reference. Slices can be used in dashboards.
8. Repeat these steps to add as many visualizations as you want.

## Updating Visualizations

You can perform update operations like edit and delete on the visualizations after you add them.

You can open the visualization in a new window/tab using the **Maximize Visualizations** icon in the visualization canvas.

### Edit Visualization

To edit a visualization:

1. On the stage that has visualizations, click the **Visualizations** tab.
2. Identify the visualization that you want to edit and click the pencil icon next to the visualization name.
3. In the **Edit Visualization** dialog box that appears, make the changes you want. You can even change the Y Axis and X Axis selections. When you change the Y Axis and X Axis values, you will notice a difference in the visualization as the basis on which the graph is plotted has changed.

### Change Orientation

Based on the data that you have in the visualization or your requirement, you can change the orientation of the visualization. You can toggle between horizontal and vertical orientations by clicking the Flip Chart Layout icon in the visualization canvas.

### Delete Visualization

You can delete the visualization if you no longer need it in the pipeline. In the visualization canvas, click the **Delete** icon to delete the visualization from the pipeline. Be careful while you delete the visualization, as it is deleted with immediate effect and there is no way to restore it once deleted.

## Working with a Live Output Table

The streaming data in the pipeline appears in a live output table.

### Hide/Unhide Columns

In the live output table, right-click columns and click **Hide** to hide that column from the output. To unhide the hidden columns, click **Columns** and then click the eye icon to make the columns visible in the output.

### Select/Unselect the Columns

Click the **Columns** link at the top of the output table to view all the columns available. Use the arrow icons to either select or unselect individual columns or all columns. Only columns you select appear in the output table.

### Pause/Restart the Table

Click **Pause/Resume** to pause or resume the streaming data in the output table.

### Perform Operations on Column Headers

Right-click on any column header to perform the following operations:

- **Hide** — hides the column from the output table. Click the Columns link and unhide the hidden columns.
- **Remove from output** — removes the column from the output table. Click the Columns link and select the columns to be included in the output table.
- **Rename** — renames the column to the specified name.
- **Function** — captures the column in Expression Builder using which you can perform various operations through the in-built functions.

### Add a Timestamp

Include timestamp in the live output table by clicking the clock icon in the output table.

### Reorder the Columns

Click and drag the column headers to right or left in the output table to reorder the columns.

## Using the Expression Builder

You can perform calculations on the data streaming in the pipeline using in-built functions of the Expression Builder.

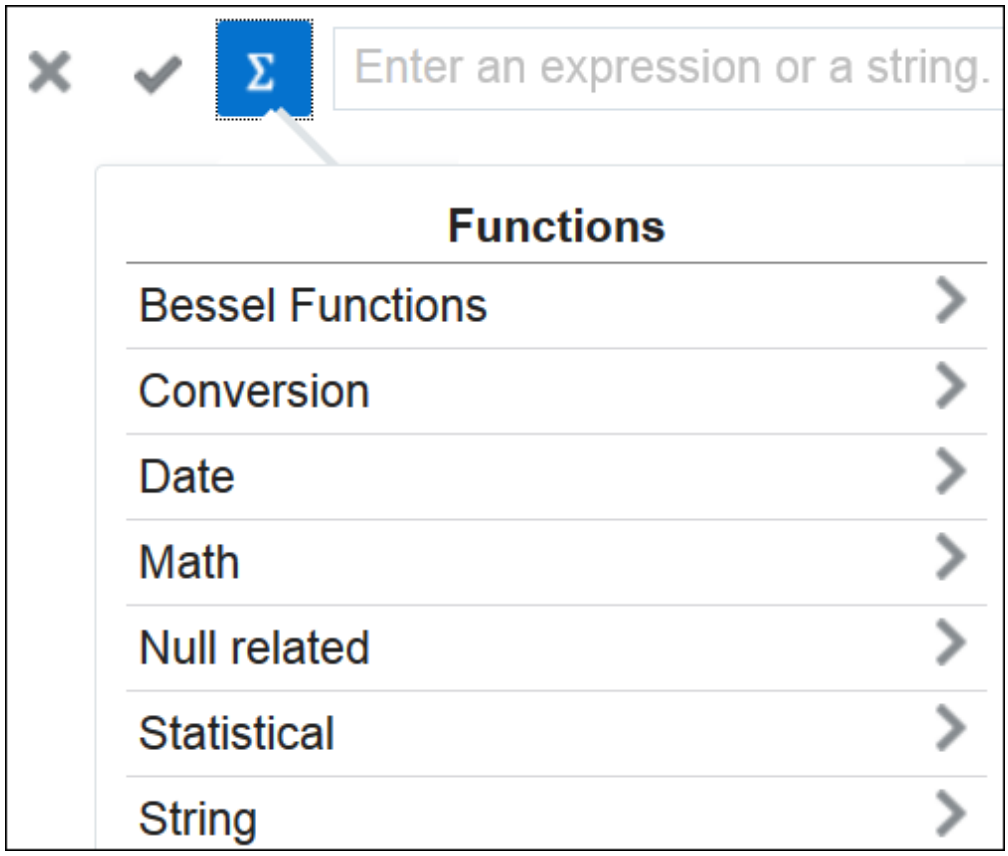
Stream Analytics supports various functions. For a list of supported functions, see Expression Builder Functions.

### Adding a Constant Value Column

A constant value is a simple string or number. No calculation is performed on a constant value. Enter a constant value directly in the expression builder to add it to the live output table.


**Using Functions**

You can select a CQL Function from the list of available functions and select the input parameters. Make sure to begin the expression with =". Click **Apply** to apply the function to the streaming data.




## Add a Pattern Stage

Patterns are templatized stages. You supply a few parameters for the template and a stage is generated based on the template.

To add a pattern stage:

1. Open a pipeline in the **Pipeline Editor**.
2. Click **Add a Stage**.
3. Select **Pattern**.
4. Choose the required pattern from the list of available patterns.
5. Enter a **Name** and **Description** for the pattern stage.

   The selected pattern stage is added to the pipeline.
6. Click **Parameters** and provide the required values for the parameters.
7. Click **Visualizations** and add the required visualizations to the pattern stage.

# Add a Rule Stage

Using a rule stage, you can add IF-THEN logic to your pipeline. A rule is a set of conditions and actions applied to a stream.

To add a rule stage:

1. Open a pipeline in the **Pipeline Editor**.
2. Click **Add a Stage**.
3. Select **Rules**.
4. Enter a **Name** and **Description** for the rule stage.
5. Click **Add a Rule**.
6. Enter **Rule Name** and **Description** for the rule and click **Done** to save the rule.
7. Select a suitable condition in the **IF** statement, **THEN** statement, and click **Add Action** to add actions within the business rules.

The rules are applied to the incoming events one by one and actions are triggered if the conditions are met.

# Add a Query Group Stage

A query group stage allows you to use more than one query group to process your data - a stream or a table in memory. A query group is a combination of summaries (aggregation functions), GROUP BYs, filters and a range window. Different query groups process your input in parallel and the results are combined in the query group stage output. You can also define input filters that process the incoming stream before the query group logic is applied, and result filters that are applied on the combined output of all query groups together.

A query group stage of the stream type applies processing logic to a stream. It is in essence similar to several parallel query stages grouped together for the sake of simplicity.

A query group stage of the table type can be added to a stream containing transactional semantic, such as a change data capture stream produced, to give just one example, by the Oracle Golden Gate Big Data plugin. The stage of this type will recreate the original database table in memory using the transactional semantics contained in the stream. You can then apply query groups to this table in memory to run real-time analytics on your transactional data without affecting the performance of your database.

## Add a Query Group: Stream

You can apply aggregate functions with different GROUP BY and window ranges to your streaming data.

To add a query group stage of type stream:

1. Open a pipeline in the **Pipeline Editor**.
2. Click the **Add Stage** button, select **Query Group** and then **Stream**.

   You can add a query stage group only at the end of the pipeline.

3. Enter a name and a description for the query group stage of the type stream and click **Save**.

   The query group stage of the type stream appears in the pipeline.

4. On the **Input Filters** tab, click **Add a Filter**. See Adding Filters to understand the steps for creating filters.

   These filters process data before it enters the query group stage. Hence, you can only see fields of the original incoming shape.

5. On the **Groups** tab, click **Add a Group**. A group can consist one or many of summaries, filters, and GROUP BYs.

   See Adding Summaries and Adding Group Bys for steps.

6. Repeat the previous step to add as many groups as you want.

7. On the **Result Filters** tab, click **Add a Filter** to filter the results.

   These filters process data before it exits the query group stage. Hence, you can see combined set of fields that get produced in the outgoing shape.

8. On the **Visualizations** tab, click **Add a Visualization** and add the required type of visualization. See Adding Visualizations for the procedure.

## Add a Query Group: Table

You can apply aggregate functions with different GROUP BYs and window ranges to a database table data recreated in memory.

To add a query group stage of the type table:

1. Open a pipeline in the **Pipeline Editor**.
2. Click the **Add Stage** button, select **Query Group** and then **Table**.
3. Enter a name and a description for the Query Group Table and click **Next**.
4. On the **Transactions Settings** screen, select a column in the **Transaction Field** drop-down list.

   The transaction column is a column from the output of the previous stage that carries the transaction semantics (insert/update/delete). Make sure that you use the values that correspond to your change data capture dataset. The default values work for Oracle GoldenGate change data capture dataset.

5. On the **Field Mappings** screen, select the columns that carry the before and after transaction values from the original database table. For example, in case of Oracle GoldenGate, the before and after values have `before_` and `after_` as prefixes, respectively. Specify a column as primary key in the table.
6. Click **Save** to create a query group stage of the type table.

   You can see the table configuration that you have specified while creating the table stage in the **Table Configuration** tab.

7. On the **Input Filters** tab, click **Add a Filter**. See Adding Filters to understand the procedure.
8. On the **Groups** tab, click **Add a Group**. A group can consist one or many of summaries, filters, and GROUP BYs.

   See Adding Summaries and Adding Group Bys for steps.

9. Repeat the previous step to add as many groups as you want.

10. On the **Result Filters** tab, click **Add a Filter** to filter the results.
11. On the **Visualizations** tab, click **Add a Visualization** and add the required type of visualization. See Adding Visualizations for the procedure.

## Configure a Target

A target defines a destination for output data coming from a pipeline.

To configure a target:

1. Open a pipeline in the **Pipeline Editor**.
2. Click **Target** in the left tree.
3. Select a target for the pipeline from the drop-down list.
4. Map each of the **Target Property** and **Output Stream Property**.

You can also directly create the target from within the pipeline editor. See Create a Target for the procedure. You can also edit an existing target.


The pipeline is configured with the specified target.

# Publish a Pipeline

You must publish a pipeline to make the pipeline available for all users of Stream Analytics and send data to targets.

A published pipeline will continue to run on your Spark cluster after you exit the Pipeline Editor, unlike the draft pipelines, which are undeployed to release resources.

To publish a pipeline:

1. Open a draft pipeline in the **Pipeline Editor**.
2. Click **Publish**.

   The Pipeline Settings dialog box opens.
3. Update any required settings.
4. Click **Publish** to publish the pipeline.

   A confirmation message appears when the pipeline is published.

# Use the Topology Viewer

*Topology* is a graphical representation and illustration of the connected entities and the dependencies between the artifacts.

The topology viewer helps you in identifying the dependencies that a selected entity has on other entities. Understanding the dependencies helps you in being cautious while deleting or undeploying an entity. Stream Analytics supports two contexts for the topology — *Immediate Family* and *Extended Family*.

You can launch the Topology viewer in any of the following ways:

- Select **Show topology** from the **Catalog Actions** menu to launch the **Topology Viewer** for the selected entity.


- Click the **Show Topology** icon in the Pipeline Editor.

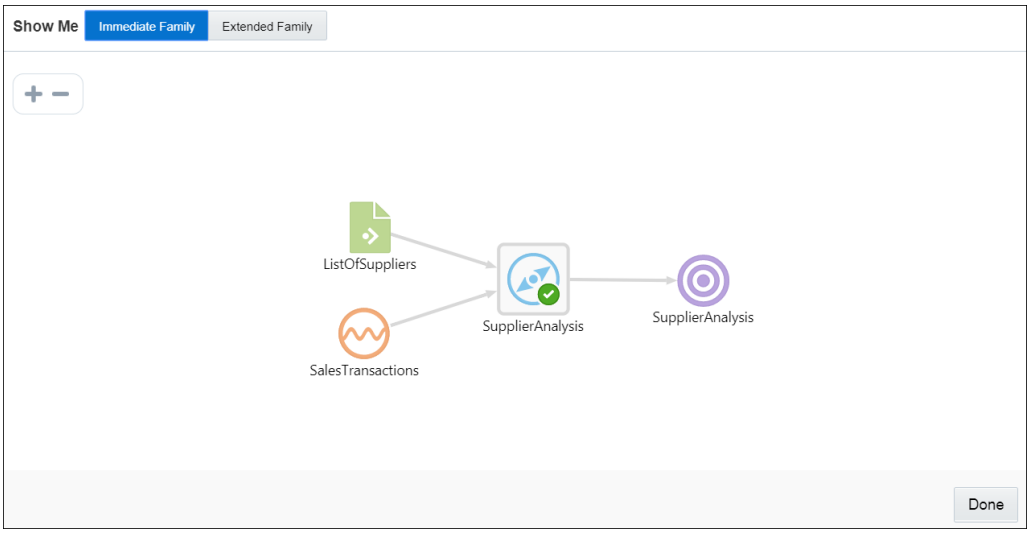

Click the **Show Topology** icon at the top-right corner of the editor to open the topology viewer.By default, the topology of the entity from which you launch the Topology Viewer is displayed. The context of this topology is **Immediate Family**, which indicates that only the immediate dependencies and connections between the entity and other entities are shown. You can switch the context of the topology to display the full topology of the entity from which you have launched the Topology Viewer. The topology in an **Extended Family** context displays all the dependencies and connections in the topology in a hierarchical manner.

> **Note:**
>
> The entity for which the topology is shown has a grey box surrounding it in the Topology Viewer.

### Immediate Family

*Immediate Family* context displays the dependencies between the selected entity and its child or parent.

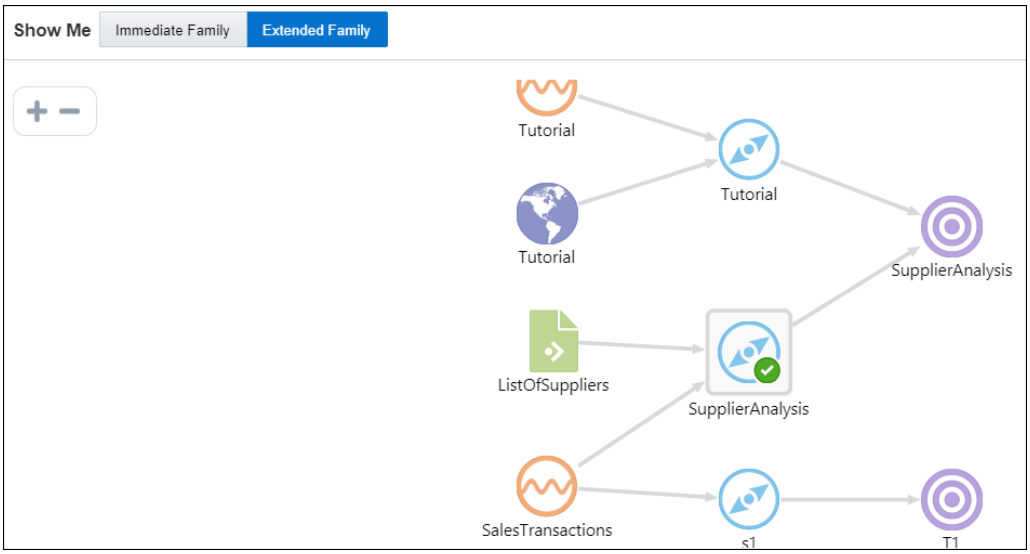
The following figure illustrates how a topology looks in the **Immediate Family**.



### Extended Family

*Extended Family* context displays the dependencies between the entities in a full context, that is if an entity has a child entity and a parent entity, and the parent entity has other dependencies, all the dependencies are shown in the Full context.

The following figure illustrates how a topology looks in the **Extended Family**.



4

# Work with Patterns

Patterns are a stage within a pipeline. When working from a pattern, you need to specify a few key fields to discover an interesting result. You can create pattern stages within the pipeline. Patterns are not stand-alone artifacts, they need to be embedded within a pipeline.

**Topics:**



## About Stream Analytics Patterns

The visual representation of the event stream varies from one pattern type to another based on the key fields you choose.

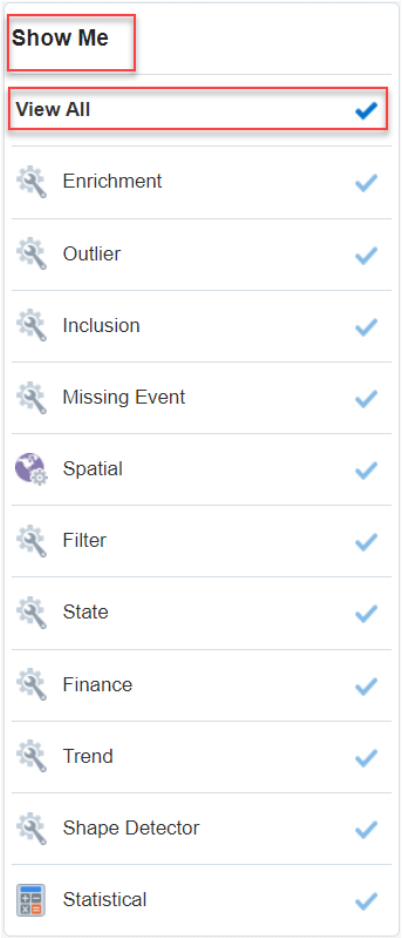
Click **Patterns** on the Home page to see all the available patterns. Use the filters at left to view different categories of pattern. You can see full descriptions and learn more about each pattern by clicking the user assistant icon. Click again to hide the extra information.

A *pattern* provides you with the results displayed in a live output stream based on common business scenarios.

> **Note:**
>
> While entering data in the fields for a specific pattern, ensure that the data you enter corresponds to the datatype of the field. If there is a mismatch between the entered data and the datatype, the pattern will not deploy and throw an error.

You can include or exclude patterns based on their categories using the **View All** link in the left panel under **Show Me**. When you click **View All**, a check mark appears next to it and all the patterns are displayed on the page.

When you want to display/view only a few/selective patterns, deselect **View All** and select the individual patterns. Only the selected patterns are shown in the catalog.




The following table lists the categories of patterns:

| Category | Pattern |
|---|---|
| Enrichment | Reverse Geo Code: Near By<br>Left Outer Join |
| Outlier | Fluctuation |
| Inclusion | Union<br>Left Outer Join |
| Missing Event | 'A' Not Followed by 'B'<br>Detect Missing Event |

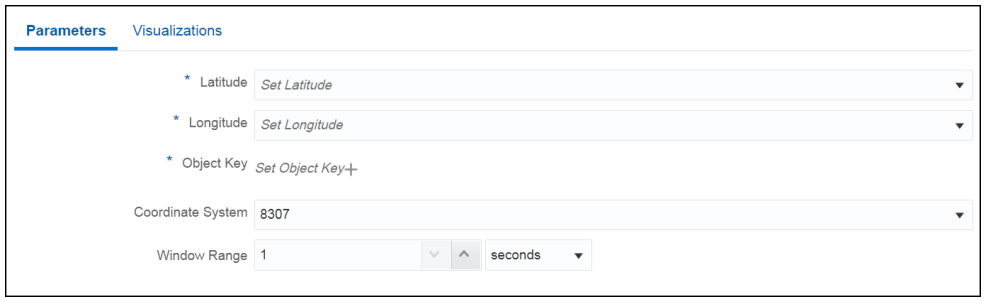
| Category | Pattern |
|---|---|
| Spatial | Proximity: Stream with Geo Fence<br>Geo Fence<br>Spatial: Speed<br>Interaction: Single Stream<br>Reverse Geo Code: Near By<br>Geo Code<br>Spatial: Point to Polygon<br>Interaction: Two Stream<br>Proximity: Two Stream<br>Direction<br>Reverse Geo Code: Near By Place<br>Proximity: Single Stream<br>Geo Filter |
| Filter | Eliminate Duplicates<br>Fluctuation |
| State | 'A' Not Followed by 'B'<br>Inverse W<br>Detect Missing Event<br>W<br>'A' Followed by 'B' |
| Finance | Inverse W<br>W |
| Trend | 'A' Not Followed by 'B<br>Top N<br>Change Detector<br>Up Trend<br>Detect Missing Event<br>Down Trend<br>'A' Followed by 'B'<br>Detect Duplicates<br>Bottom N |
| Shape Detector | Inverse W<br>W |
| Statistical | Correlation<br>Quantile |

## About the Spatial: Speed Pattern

Use this pattern to get the output average speed over the selected window range of a moving object.

For example, to analyze the average speed of a car.

Make sure that you have set the proxy details in System Settings.




Provide suitable values for the following parameters:

- **Latitude**: the latitude of the location. Select a suitable value.
- **Longitude**: the longitude of the location. Select a suitable value.
- **Object Key**: select a suitable value for the object key.
- **Coordinate System**: the default value is 8307 and this is the only value supported.
- **Window Range**: a time range over which the speed is being calculated for an event. For example, if `window range=5 seconds` and object key is `phone no.`, then all the events with same `phone no.` received over last 5 seconds are used to calculate the average speed of that event.

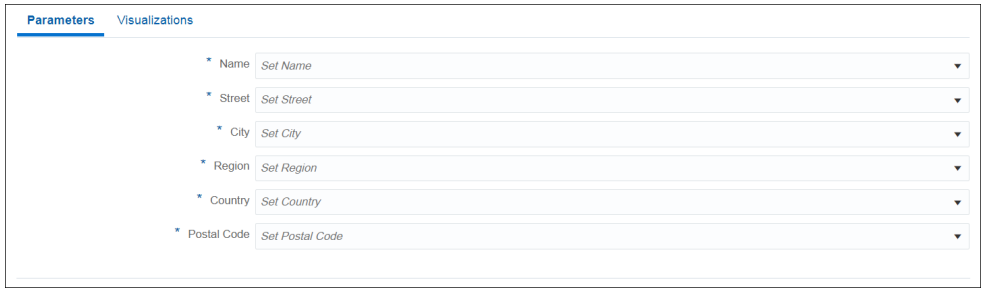
The outgoing shape contains **speed** as an added field along with the incoming fields. This is a numeric field, but the **speed** is measured in `meters per second`.

# About the Geo Code Pattern

When analyzing data, you may encounter situations where you need to obtain the geographical code of a moving object based.

Use this pattern to get geographic coordinates (like latitude and longitude) for an address.

Make sure that you have set the proxy details in System Settings.




Provide suitable values for the following parameters:

- **Name** — select a suitable value that can be uses as the place name. This is a mandatory parameter.
- **Street** — select a suitable value that can be used as the street name. This is a mandatory parameter.
- **City** — select a suitable value that can be used as the city name. This is a mandatory parameter.

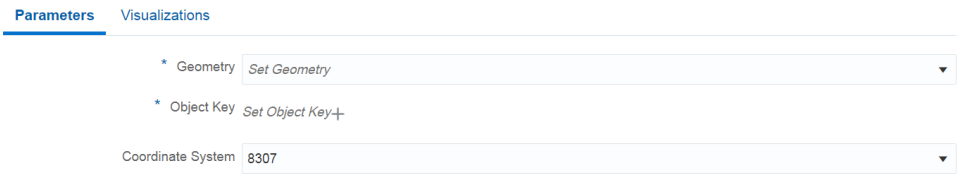
- **Region** — select a suitable value that can be used for the region. This is a mandatory parameter.
- **Country** — select a suitable value for the country. This is a mandatory parameter.
- **Postal Code** — select a suitable value for the postal code. This is a mandatory parameter.

The outgoing shape contains latitude and longitude as additional fields along with incoming fields.

## About the Interaction: Single Stream Pattern

Use this pattern to get interaction of an object with every other object in a stream.

For example, you can see if a set of sailing ships are too close to each other.




Provide suitable values for the following parameters:

- **Geometry**: the field of type `SDO_GEOMETRY` data type. Only field of `SDO_GEOMETRY` data type should be chosen for such fields.
- **Object Key**: field used to uniquely identify object on the geo fence and is used for partitioning of data where supported.
- **Coordinate System**: the default value is 8307 and this is the only value supported.

The outgoing shape contains two more fields along with the incoming shape: `isInteract` and `distance`. `isInteract` is `true`if two shapes interact with each other, i.e., any or some portion of the two objects overlap. `distance` between them is `0`, if no overlapping is observed; `isInteract` is `false` and `distance` is shown between those two objects as a positive number.

## About the Interaction: Two Stream Pattern

Two shapes are said to interact with each other if any part of the shape overlaps. If two shapes interact, the distance between them is *zero*.

Use this pattern to get interaction of an object in one stream with objects in another stream.


Provide suitable values for the following parameters:

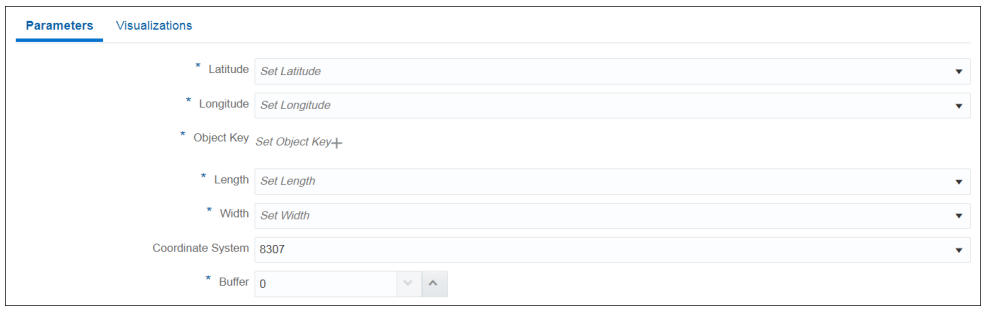
- **Geometry**: select a suitable value for geometry. This is a mandatory value.
- **Object Key**: select a suitable value for the object key. This is a mandatory value.
- **Event Stream 2**: select the second event stream. This is a mandatory value.
- **Geometry**: select a suitable value for geometry within the second stream. This is a mandatory value.
- **Object Key**: select a suitable value for the object key within the second stream. This is a mandatory value.
- **Coordinate System**: the default value is 8307 and this is the only value supported.

The outgoing shape contains two additional fields along with the incoming shape: `isInteract` and `distance`. `isInteract` is `true`if two shapes interact with each other, i.e., any or some portion of the two objects overlap. `distance` between them is `0`, if no overlapping is observed; `isInteract` is `false` and `distance` is shown between those two objects as a positive number.

# About the Spatial: Point to Polygon Pattern

Use this pattern to get an object shape based on geographical coordinates, fixed length and breadth of and object.

For example, if you know the length and breadth of a group of a fleet of ships, you can get the shape of a ship using the position coordinates, where the coordinates keep changing as the ship moves.




Provide suitable values for the following parameters:

- **Latitude**: select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude. This is a mandatory parameter.

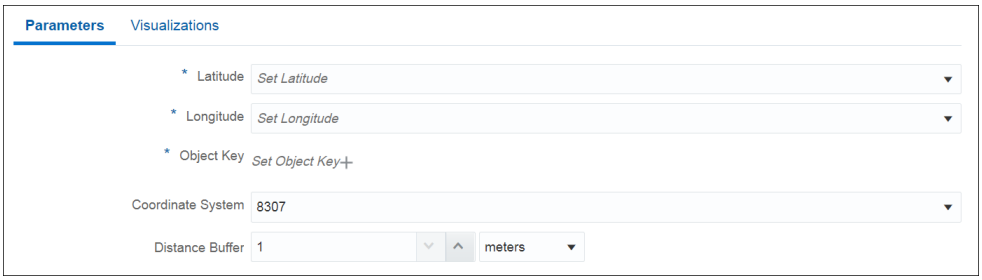
- **Object Key**: select a suitable value for the object key. This is a mandatory parameter.
- **Length**: select a suitable field to be used as the reference length of the polygon.
- **Width**: select a suitable field to be used as the reference width of the polygon.
- **Coordinate System**: the default value is 8307 and this is the only value supported.
- **Buffer**: enter a positive value to be used as the geometry buffer.

The outgoing shape contains **Updatedshape** as an additional attribute which is the updated point into geometry of type `SDO_GEOMETRY` along with the incoming shape.

## About the Proximity: Single Stream Pattern

Use this pattern to get proximity of each object with every other object in a stream.

For example, if there is stream of flying airplanes and the distance buffer is 1000 meters. You can raise an alert as the two planes come into a proximity of 1000 meters or less.




Provide suitable values for the following parameters:

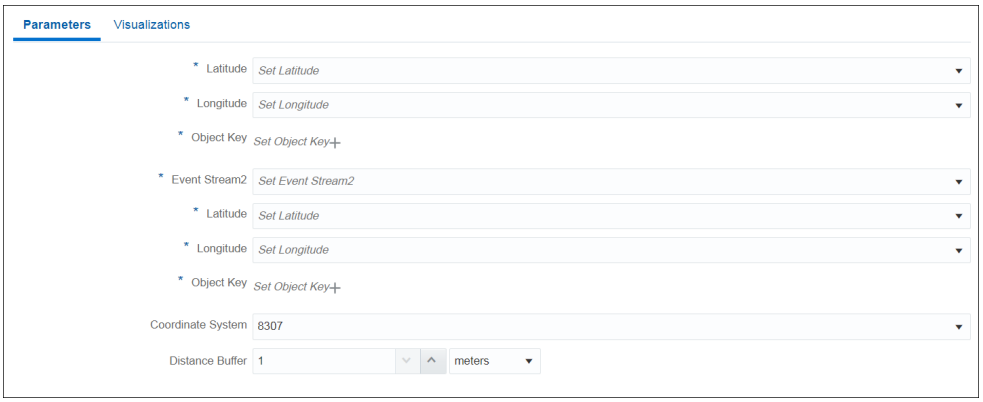
- **Latitude**: select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key**: select a suitable value for the object key. This is a mandatory parameter.
- **Coordinate System**: the default value is 8307 and this is the only value supported.
- **Distance Buffer**: enter a proximity value for the distance buffer. This is the distance that two points can be apart and still be considered the same. Select an appropriate unit for the distance.

The outgoing shape displays **distance** as another column, which is the distance between two object under consideration along with the incoming shape.

## About the Proximity: Two Stream Pattern

Use use this pattern to get the proximity between objects of two streams.

The distance buffer acts as a filter in this pattern stage. For example, if there is a driver and passenger stream, you can get the proximity of each passenger with every other driver using a filter criteria of ‘within a distance of 1 km’.




Provide suitable values for the following parameters:

- **Latitude**: select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key**: select a suitable value for the object key. This is a mandatory value.
- **Event Stream 2**: select the second event stream. This is a mandatory value.
- **Latitude**: select a suitable value for the latitude in the second stream. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude in the second stream. This is a mandatory parameter.
- **Object Key**: select a suitable value for the object key in the second stream. This is a mandatory parameter.
- **Coordinate System**: the default value is 8307 and this is the only value supported.
- **Distance Buffer**: enter a proximity value for the distance buffer. This field acts as a filter criteria of two objects and the objects that do not fall in this distance (distance between them is more than chosen distance buffer) are filtered from result set.

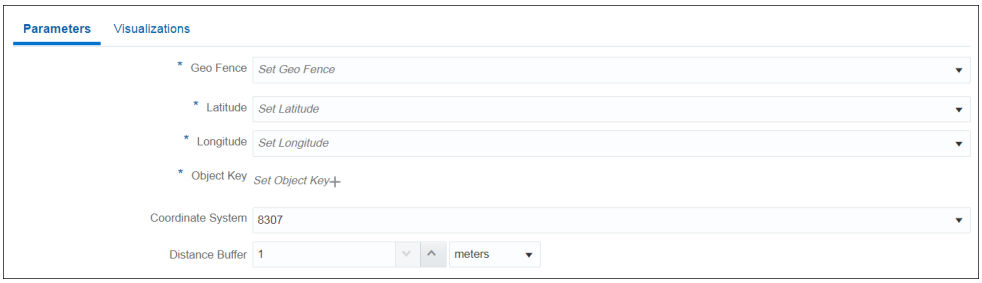
> **Note:**
>
> When a pipeline with this pattern has a database reference with cache enabled, the pattern does not display any output in the live output stream.

The outgoing shape displays **distance** as another column, which is the distance between two object under consideration along with the incoming shape.

## About the Proximity: Stream with Geo Fence Pattern

Use this pattern to get proximity of an object with a virtual boundary or geo fence.

For example, if you have certain stores in the city of California, you can send promotional messages as soon as the customer comes into a proximity of 1000 meters from any of the stores.




Provide suitable values for the following parameters:

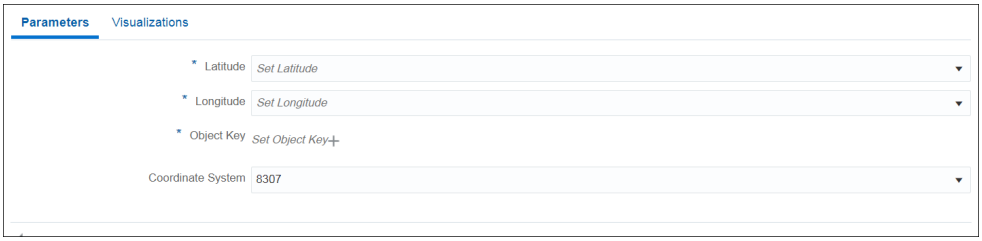
- **Geo Fence**: select a geo fence that you like to analyze.
- **Latitude**: select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key**: select a suitable value for the object key. This is a mandatory parameter.
- **Coordinate System**: the default value is 8307 and this is the only value supported.
- **Distance Buffer**: enter a proximity value for the distance buffer. This field acts as a filter criteria for events and the events that do not fall in this distance (distance between them is more than chosen distance buffer) are filtered from result set.

The outgoing shape displays **distance** as another column, which is the distance between the object and geo fence under consideration along with the incoming shape.

## About the Direction Pattern

Use this pattern to get the direction of a moving object.

For example, you can evaluate the direction of a moving truck.




Provide suitable values for the following parameters:

- **Latitude**: select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude**: select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key**: select a suitable value for the object key. This is a mandatory parameter.
- **Coordinate System**: the default value is 8307 and this is the only value supported.

The outgoing shape displays **direction** as one of the columns, which is of type `String` along with the incoming shape.

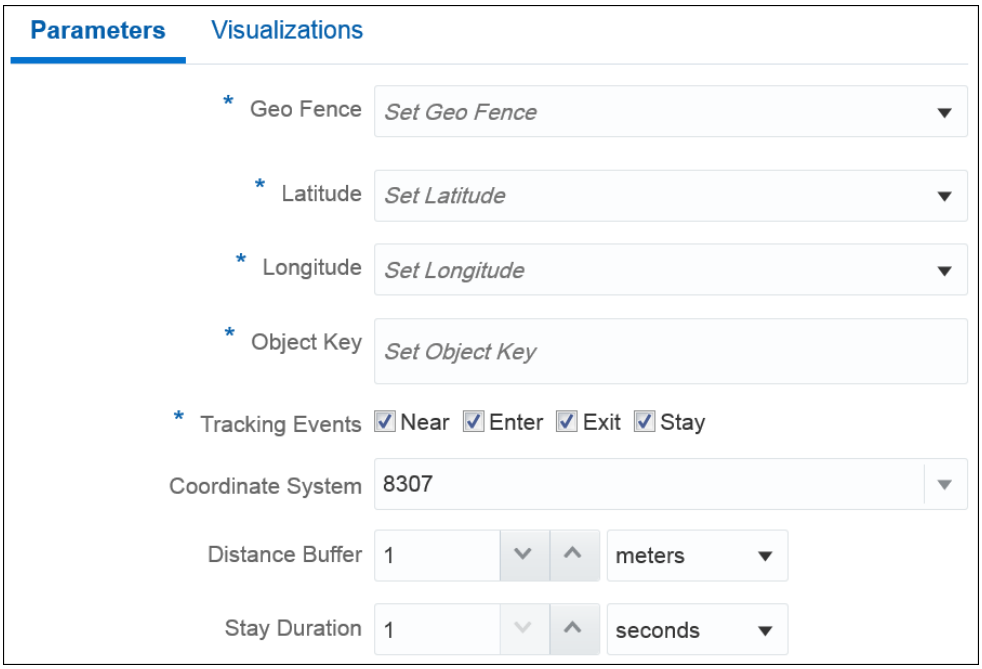
# About the Geo Fence Pattern

Use this pattern when you want to track object relation with a virtual boundary called goe fence.

Relations can be `Enter`, `Exit`, `Stay`, or `Near` with respect to a geo fence. For example, you can trigger an alert when an object enters the geo fence. You can also analyze a stream containing geo-location data. It helps in determining how events are related to a polygon in a geo fence.

The geo-location can be:

- Near to Geo Fence
- Exiting Geo Fence
- Staying within Geo Fence for a specified duration
- Entering Geo Fence




Provide suitable values for the following parameters:

- Geo Fence
- Latitude
- Longitude
- Object Key
- Tracking Events
  - Near
  - Enter
  - Exit
  - Stay

- Coordinate system
- Distance Buffer — this parameter is enabled only if you select **Near** option in **Tracking Events**. This field acts as a buffer for filtering results. Only those events or objects which are within the specified distance from the geo fence are displayed in events table with status as `Near`.
- Stay Duration — this parameter is enabled only if you select **Stay** in **Tracking Events**. You can specify the stay duration and this duration acts as a filter for objects inside the geo fence. If an object stays for a duration more than the specified duration, only then the events are considered, else events are filtered out.

The outgoing shape displays **Status** and **PlaceName** as two extra columns in the output along with the incoming shape, where **Status** is one of `Enter`, `Exit`, `Stay`, or `Near` based on how the object behaves with geo fence. **PlaceName** is the name of geo fence with which status is being evaluated.

## About the Geo Fence Filter: Inside Pattern

Use this pattern to when you want to track objects inside a virtual boundary.

For example, if users move from one geographical location to another, you can send promotional messages to the users when they are inside a specified geo fence.


Provide suitable values for the following parameters:

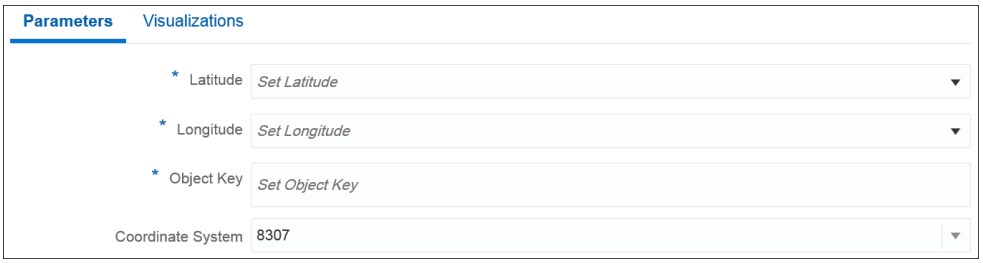
- **Geo Fence** — select one of the existing geo fences to analyze. This is a mandatory field.
- **Latitude** — select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude** — select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key** — select a suitable value for the object key. This field acts as a partitioning criteria and also used to uniquely identify objects. This is a mandatory parameter.
- **Coordinate system** — the default value is 8307 and this is the only value supported.

The outgoing shape displays **Status** and **PlaceName** as two extra columns in the output along with the incoming shape, where **Status** is `Inside` if the object is inside geo fence (else the event is not considered) and **PlaceName** is the name of geo fence with which status is being evaluated.

# About the Reverse Geo Code: Near By Pattern

Use this to obtain nearest place for the specified geographical coordinates.

Make sure that you have set the proxy details in System Settings.




Provide suitable values for the following parameters:

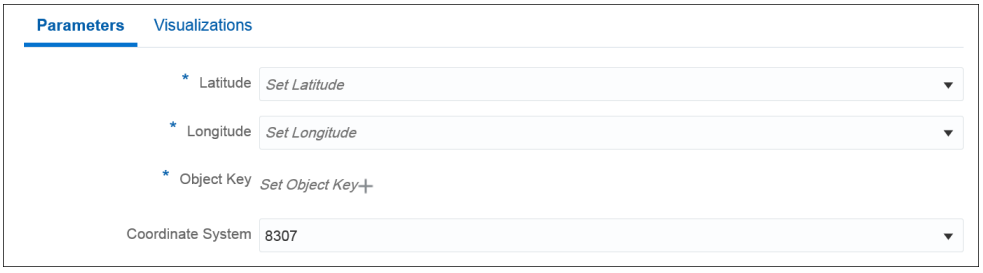
- **Latitude** — select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude** — select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key** — select a suitable value for the object key. This is a mandatory parameter.
- **Coordinate system** — the default value is 8307 and this is the only value supported.

The outgoing shape displays **PlaceName** as an additional column along with the incoming shape. This column is the nearest place for specified longitude and latitude.

# About the Reverse Geo Code: Near By Place Pattern

Use this pattern to obtain the near by location with granular information like city, country, street etc. for the specified latitude and longitude.

Make sure that you have set the proxy details in System Settings.



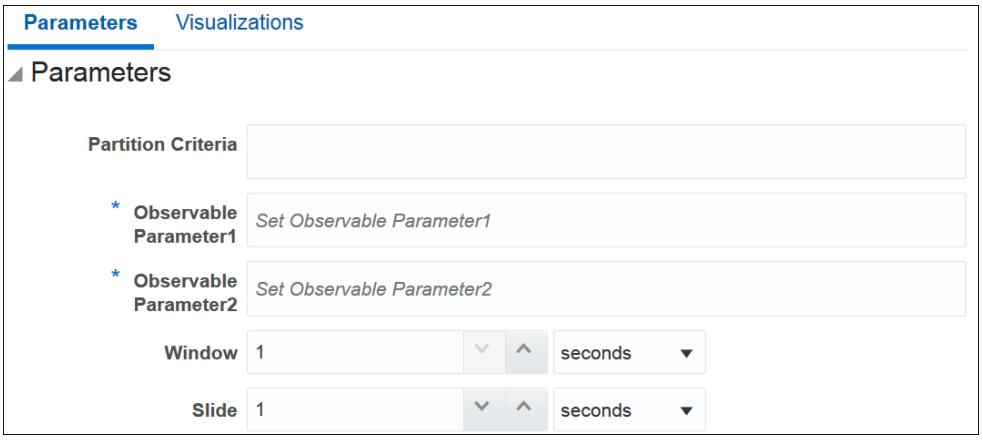

Provide suitable values for the following parameters:

- **Latitude** — select a suitable value for the latitude. This is a mandatory parameter.
- **Longitude** — select a suitable value for the longitude. This is a mandatory parameter.
- **Object Key** — select a suitable value for the object key. This is a mandatory parameter.
- **Coordinate system** — the default value is 8307 and this is the only value supported.

The outgoing shape displays **PlaceName** as an additional column along with the incoming shape. This column is the nearest place for specified longitude and latitude.

## About the Correlation Pattern

Use this pattern if you need to identify correlation between two numeric parameters.



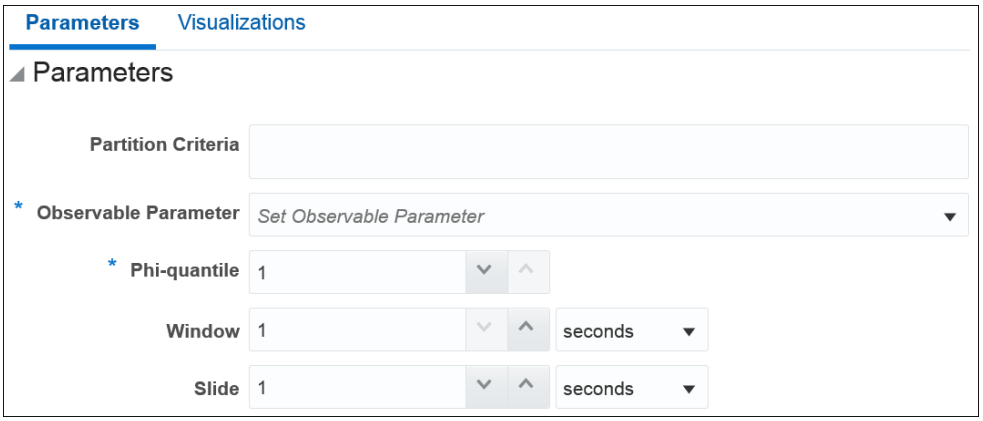

Provide suitable values for the following parameters:

- **Partition Criteria**: the field by which you want to partition.
- **Observable Parameter 1**: the first value used to identify the correlation.
- **Observable Parameter 2**: the second value used to identify the correlation.
- **Window**: a rolling time period, the duration within which the correlation is identified.
- **Slide**: the frequency at which you want to refresh the data.

The outgoing shape is same as the incoming shape.

## About the Quantile Pattern

Use this pattern if you need to calculate the value of quantile function.




Provide suitable values for the following parameters:

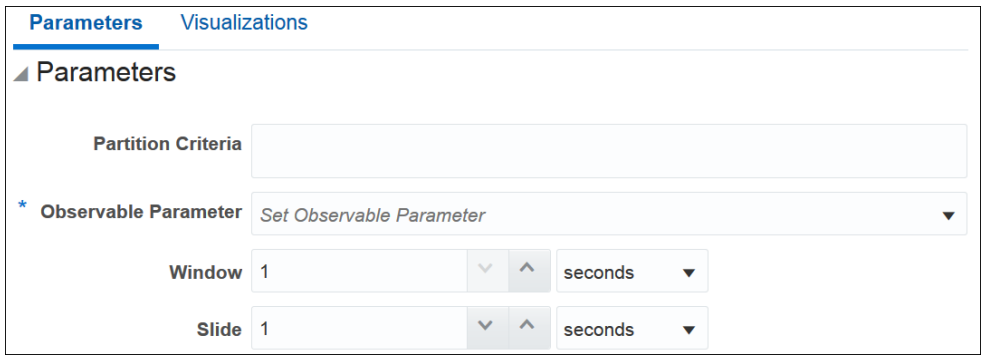
- **Partition Criteria**: the field based on which you want to partition.

- **Observable Parameter**: the observable to calculate the quantile.
- **Phi-quantile**: the value is used to calculate the quantile of the selected event stream.
- **Window**: a rolling time period, within which the events will be collected and quantile is calculated.
- **Slide**: how frequent newly updated output will be pushed downstream and into the browser.

The outgoing shape is the same as the incoming shape.

## About the Standard Deviation Pattern

Use this pattern to calculate the standard deviation of the selected values with the expected values.



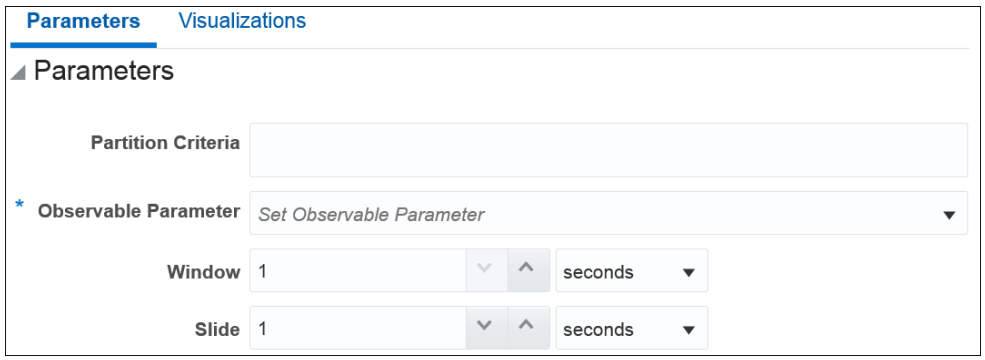

Provide suitable values for the following parameters:

- **Partition Criteria**: the field based on which you want to partition.
- **Observable Parameter** : the value used to identify the standard deviation.
- **Window**: a rolling time period, the duration within which the standard deviation is identified.
- **Slide**: the frequency at which you want to refresh the data.

The outgoing shape is the same as the incoming shape.

## About the Median Pattern

Use this pattern to calculate the median of an event stream with respect to a specific parameter.



Provide suitable values for the following parameters:

- **Partition Criteria**: the field based on which you want to partition.
- **Observable Parameter**: the observable to calculate the median.
- **Window**: a rolling time period, within which the events will be collected and median is calculated.
- **Slide**: how frequent newly updated output will be pushed downstream and into the browser.

The outgoing shape is same as the incoming shape.

## About the Detect Duplicates Pattern

The Detect Duplicates pattern detects duplicate events in your stream according to the criteria you specify and within a specified time window. Events may be partially or fully equivalent to be considered duplicates.

Use this pattern to understand how many duplicate events your stream has. For example, when you suspect that your aggregates are offset, you may want to check your stream for duplicate events.


Provide suitable values for the following parameters:

- **Duplicate Criteria**: a list of fields, whose values will be compared between events to look for identical values. If all the configured fields have identical values, the incoming event will be considered a duplicate and an outgoing event will be fired.
- **Window**: a time period, within which the duplicates will be searched for. For example, if you set the window to 10 seconds, a duplicate event that arrives 9 seconds after the first one will be trigger an outgoing event, while a duplicate event that arrives 11 seconds after the first one will not do so.

### Outgoing Shape

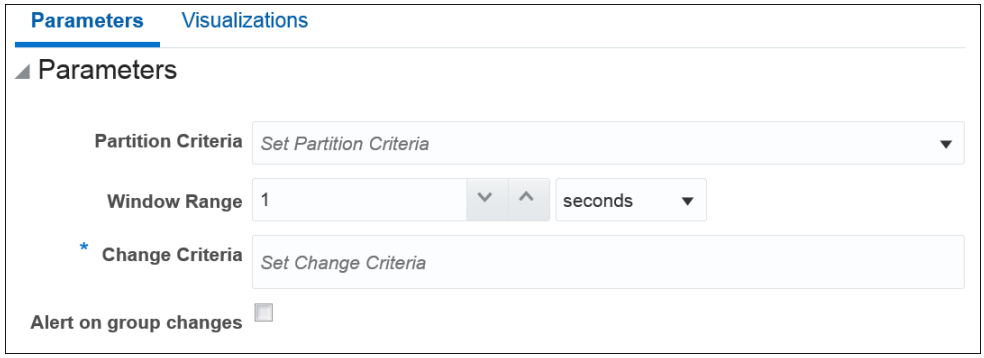
The outgoing shape is the same as the incoming shape with one extra field: `Number_of_Duplicates`. This extra field will carry the number of duplicate events that have been discovered. All the other fields will have values of the last duplicate event.

## About the Change Detector Pattern

The Change Detector pattern looks for changes in the values of your event fields and report the changes once they occur within a specified range window. For example, and events arrives with value `value1` for field `field1`. If any of the following incoming events within a specified range window contains a value different from `value1`, an alert is triggered. You can designate more than one field to look for changes.

Use it when you need to be aware of changes in a normally stable value. For example, a sensor reading that is supposed to be the same for certain periods of time and changes in readings may indicate issues.

The default configuration of this pattern stage is to alert on change of any selected fields.




Provide suitable values for the following parameters:

- **Partition Criteria**: a field to partition your stream by. For example, your stream contains events issues by a number of sensors. All sensors send the same but individual data. You would want to compare readings of a sensor to previous readings of the same sensor and not just a previous event in your stream, which is very likely to be from a different sensor. Select a field that would uniquely identify your sensors, such as sensor Id. This field is optional. For example, if your stream contains readings from just one sensor, you do not need to partition your data.
- **Window range**: a time period, within which the values of designated fields are compared for changes. For example, if you set the window range to 10 seconds, an event with changes in observed fields will trigger an alert if it arrives within 10 seconds after the initial event. The clock starts at the initial event.
- **Change Criteria**: a list of fields, whose values will be compared between events to look for changes. If the fields contain no changes, no alerts will be generated.
- **Alert on group changes**: this parameter is responsible for the default group changes support. If it is `OFF`, then alert on at least one field changes. If it is `ON`, then sends alert on every field change.

### Outgoing Shape

The outgoing shape is based on the incoming shape, the difference being that all the fields except the one in the partition criteria parameter will be duplicated to carry both the initial event values and the change event values. Let's look at an example. Your incoming event contains the following fields:

- `sensor_id`
- `temperature`
- `pressure`
- `location`

Normally, you would use sensor_id to partition your data and say you want to look for changes in temperature. So, select sensor_id in the partition criteria parameter and temperature in the change criteria parameter. Use a range window that fits your use case. In this scenario, you will have the following outgoing shape:

- `sensor_id`
- `temperature`
- `orig_temperature`
- `pressure`
- `orig_pressure`
- `location`
- `orig_location`

The *orig_* fields carry values from the initial event. In this scenario, `temperature` and `orig_temperature` values are different, while pressure and `orig_pressure`, `location`, and `orig_location` may have identical values.

## About the W Pattern

The W pattern, also known as a double bottom chart pattern, is used in the technical analysis of financial trading markets.

Use this pattern to detect when an event data field value rises and falls in “W” fashion over a specified time window. For example, use this pattern when monitoring a market data feed stock price movement to determine a buy/sell/hold evaluation.


Provide suitable values for the following parameters:

- **Partition Criteria**: a field to partition your stream by. For example, a ticker symbol.
- **Window**: a time period, within which the values of the designated field are analyzed for the W shape.
- **Tracking value**: a field, whose values are analyzed for the W shape.

### Outgoing Shape

The outgoing shape is based on the incoming shape with an addition of five new fields. The new fields are:

- `firstW`
- `firstValleyW`
- `headW`
- `secondValleyW`
- `lastW`

The new fields correspond to the tracking value terminal points of the W shape discovered in the feed. The original fields correspond to the last event in the W pattern.

## Rule

A *rule* is a set of conditions applied to the incoming stream and a set of actions performed on the stream when conditions are true. Each event is analyzed independently of other events.

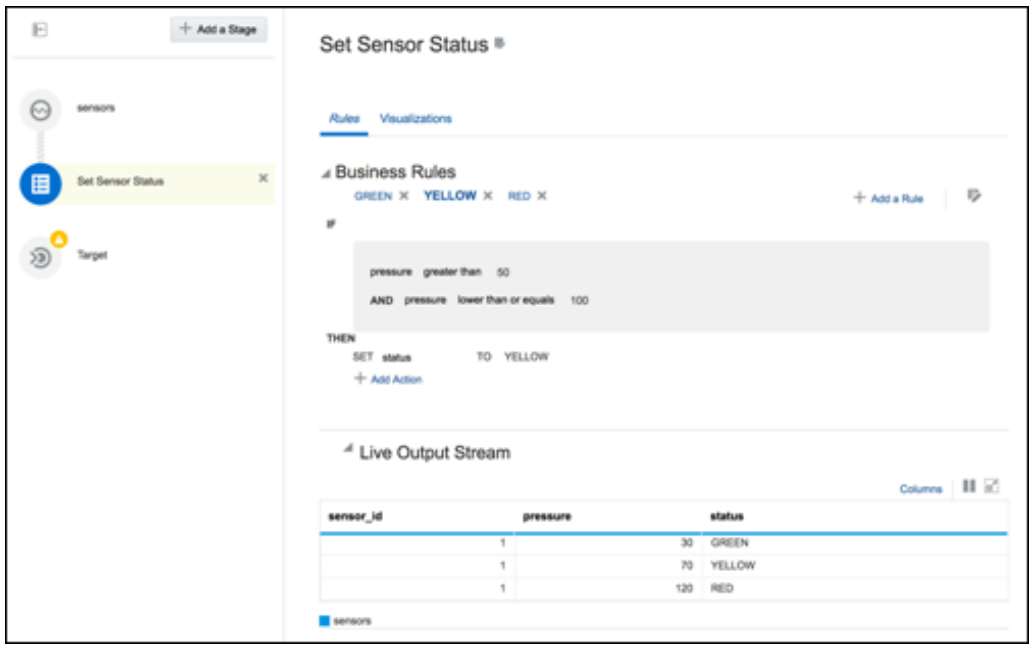
For example, assume that your stream is a stream from pressure sensors and has the following fields:

- `sensor_id`
- `pressure`
- `status`

If you want to assign a status value based on the pressure, you can define the following rules:

- if the pressure is less than or equal to 50, the status must be set to GREEN
- if the pressure is between 50 and 100, the status must be set to YELLOW
- if the pressure is greater than 100, the status must be set to RED.

To achieve this, you need to create these rules in a rule stage. The YELLOW rule for example, looks as shown below:




The rules get applied to the events sequentially and actions are triggered if the conditions are met. If you look at the data in the previous screen, the pressure value is 120 in the last row and hence the RED rule conditions resolve to *true*.

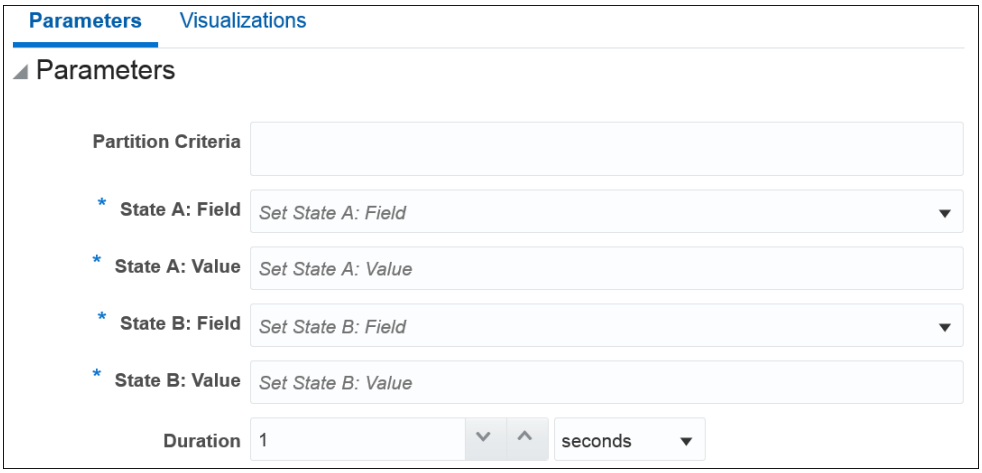
You must be careful while defining the rules. Logical loops or contradictory rules lead to the application never returning any outgoing events. For example, the following rules force the application into running forever without any outgoing events:

- Rule 1: if n > 0, set n to -1
- Rule 2: if n <=0, set n to 1

## About the ‘A’ Followed by ‘B’ Pattern

The 'A' Followed by 'B' pattern looks for particular events following one another and will output an event when the specified sequence of events occurs.

Use it when you need to be aware of a certain succession of events happening in you flow. For example, if an order status `BOOKED` is followed by an order status `SHIPPED` (skipping status `PAID`), you need to raise an alert.




Provide suitable values for the following parameters:

- **Partition Criteria**: (Optional) a field to partition your stream by. In the order example above, it may be `order_id`.
- **State A: field**: an initial state field, whose value will be used in the comparison of two events. In our example, it will be `order_status`.
- **State A: value**: the initial field state value. In our example, `BOOKED`.
- **State B: field**: a consecutive state field, whose value will be used in the comparison of two events. In our example, it will be `order_status` again.
- **State B: value**: the consecutive field state value. In our example, `SHIPPED`.
- **Duration**: the time period, within which to look for state changes.

### Outgoing Shape

The outgoing shape is based on the incoming shape. A new `abInterval` field is added to carry the value of the time interval between the states in nanosecond. Also, all but the partition criteria fields are duplicated to carry values from both a and b states. For example, if you have the following incoming shape:

- `order_id`
- `order_status`
- `order_revenue`

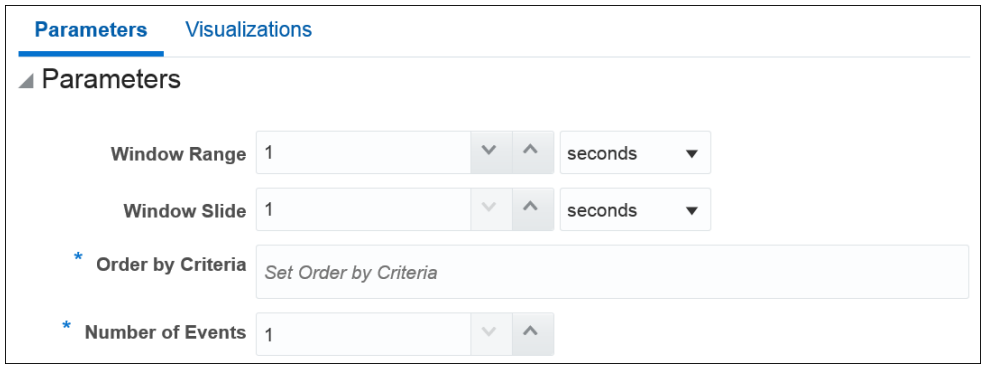
You will get the following outgoing shape:

- `order_id`
- `abInterval`
- `order_status` (this is the value by which you partition your stream)
- `aState_order_status` (this is the value of order_status in state A, in our example 'BOOKED')
- `order_revenue` (this is the value of order_revenue in state B)
- `aState_order_revenue` (this is the value of order_revenue in state A)

## About the Top N Pattern

The Top N pattern will output N events with highest values from a collection of events arriving within a specified time window sorted not in the default order of arrival but the way you specify.

Use it to get the highest values of fields in your stream within a specified time window. For example, use it to get N highest values of pressure sensor readings.




Provide suitable values for the following parameters:

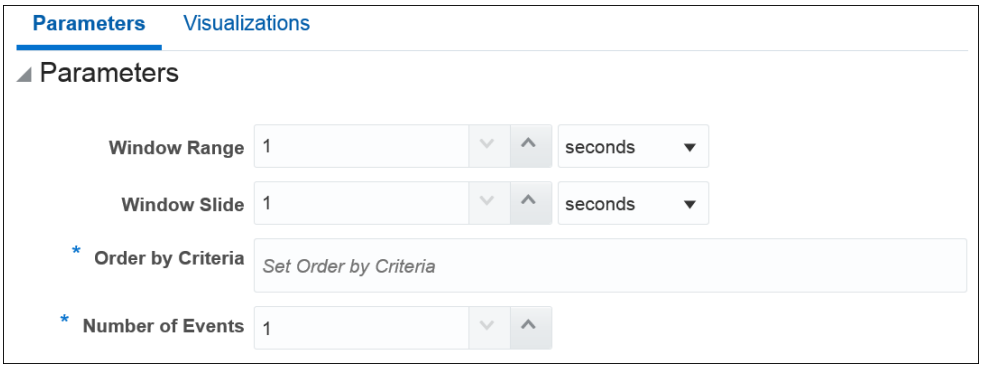
- **Window Range**: a rolling time period, within which the events will be collected and ordered per your ordering criteria.
- **Window Slide**: how frequent newly updated output will be pushed downstream and into the browser.
- **Order by Criteria**: a list of fields to use to order the collection of events.
- **Number of Events**: a number of top value events to output.

The outgoing shape is the same as the incoming shape.

## About the Bottom N Pattern

The Bottom N pattern will output N events with lowest values from a collection of events arriving within a specified time window sorted not in the default order of arrival but the way you specify.

Use it to get the lowest values of fields in your stream within a specified time window. For example, use it to get N lowest values of pressure sensor readings.




Provide suitable values for the following parameters:

- **Window Range**: a rolling time period, within which the events will be collected and ordered per your ordering criteria.
- **Window Slide**: how frequent newly updated output will be pushed downstream and into the browser.
- **Order by Criteria**: a list of fields to use to order the collection of events.
- **Number of Events**: a number of bottom value events to output.

The outgoing shape is the same as the incoming shape.

## About the Up Trend Pattern

The Up Trend pattern detects a situation when a numeric value goes invariably up over a period of time.

Use the pattern if you need to detect situations of a constant increase in one of your numeric values. For example, detect a constant increase in pressure from one of your sensors.


Provide suitable values for the following parameters:

- **Partition Criteria**: a field by which to partition your stream. For example, your stream contains events issues by a number of sensors. All sensors send the same but individual data. You would want to compare readings of a sensor to previous readings of the same sensor and not just a previous event in your stream, which is very likely to be from a different sensor. Select a field that would uniquely identify your sensors, such as sensor id. This field is optional. For example, if your stream contains readings from just one sensor, you do not need to partition your data.
- **Duration**: a time period, within which the values of the designated field are analyzed for the upward trend.

- **Tracking value**: a field, whose values are analyzed for the upward trend.

### Outgoing Shape

The outgoing shape is based on the incoming shape with an addition of two new fields. For example, if your incoming event contains the following fields:

- `sensor_id`
- `temperature`
- `pressure`
- `location`

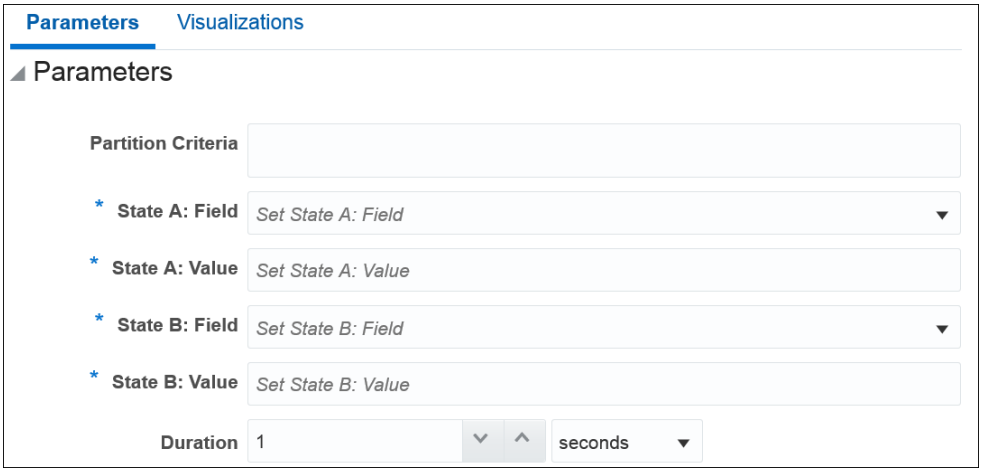
Normally, you would use `sensor_id` to partition your data and say you want to look for the upward trend in temperature. So, select `sensor_id` in the partition criteria parameter and temperature in the tracking value parameter. Use a duration that fits your use case. In this scenario, you will have the following outgoing shape:

- `sensor_id`
- `startValue` (this is the value of temperature that starts the trend)
- `endValue` (this is the value of temperature that ends the trend)
- `temperature` (the value of the last event)
- `pressure` (the value of the last event)
- `location` (the value of the last event)

## About the ‘A’ Not Followed by ‘B’ Pattern

The 'A' Not Followed by 'B' pattern will look for a missing second event in a particular combination of events and will output the first event when the expected second event does not arrive within the specified time period.

Use it when you need to be aware of a specific event not following its predecessor in your flow. For example, if an order status `BOOKED` is not followed by an order status `PAID` within a certain time period, you may need to raise an alert.




Provide suitable values for the following parameters:

- **Partition Criteria**: (Optional) a field to partition your stream by. In the order example above, it may be `order_id`.
- **State A: field**: an initial state field, whose value will be used in the comparison of two events. In our example, it will be `order_status`.
- **State A: value**: the initial field state value. In our example, `BOOKED`.
- **State B: field**: a consecutive state field, whose value will be used in the comparison of two events. In our example, it will be `order_status` again.
- **State B: value**: the consecutive field state value. In our example, `SHIPPED`.
- **Duration**: the time period, within which to look for state changes.

### Outgoing Shape

The outgoing shape is the same as incoming shape. If the second (state B) event does not arrive within the specified time window, the first (state A) event is pushed to the output.

## About the Down Trend Pattern

The Down Trend pattern detects a situation when a numeric value goes invariably down over a period of time.

Use this pattern if you need to detect situations of a constant reduction in one of your numeric values. For example, detect a constant drop in pressure from one of your sensors.


Provide suitable values for the following parameters:

- **Partition Criteria**: a field to partition your stream by. For example, your stream contains events issues by a number of sensors. All sensors send the same but individual data. You would want to compare readings of a sensor to previous readings of the same sensor and not just a previous event in your stream, which is very likely to be from a different sensor. Select a field that would uniquely identify your sensors, such as sensor id. This field is optional. For example, if your stream contains readings from just one sensor, you do not need to partition your data.
- **Duration**: a time period, within which the values of the designated field are analyzed for the downward trend.
- **Tracking value**: a field, whose values are analyzed for downward trend.

### Outgoing Shape

The outgoing shape is based on the incoming shape with an addition of two new fields. Let's look at an example. Your incoming event contains the following fields:

- `sensor_id`
- `temperature`
- `pressure`
- `location`

Normally, you would use `sensor_id` to partition your data and say you want to look for the downward trend in temperature. So, select `sensor_id` in the partition criteria parameter and temperature in the tracking value parameter. Use a duration that fits your use case. In this scenario, you will have the following outgoing shape:

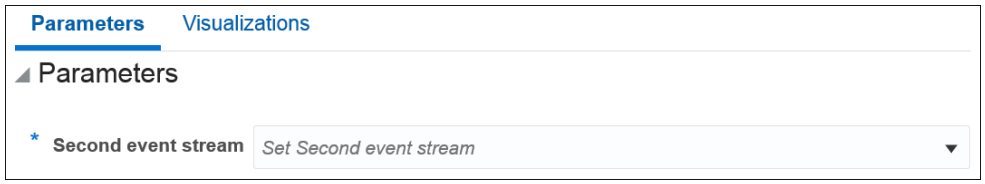
- `sensor_id`
- `startValue` (this is the value of temperature that starts the trend)
- `endValue` (this is the value of temperature that ends the trend)
- `temperature` (the value of the last event)
- `pressure` (the value of the last event)
- `location` (the value of the last event)

The pattern is visually represented based on the data you have entered/selected.

## About the Union Pattern

The Union pattern merges two streams with identical shapes into one.

Use this pattern if you have two streams with identical shapes that you want to merge into one, for example when you have two similar sensors sending data into two different streams, and you want to process the streams simultaneously, in one pipeline.



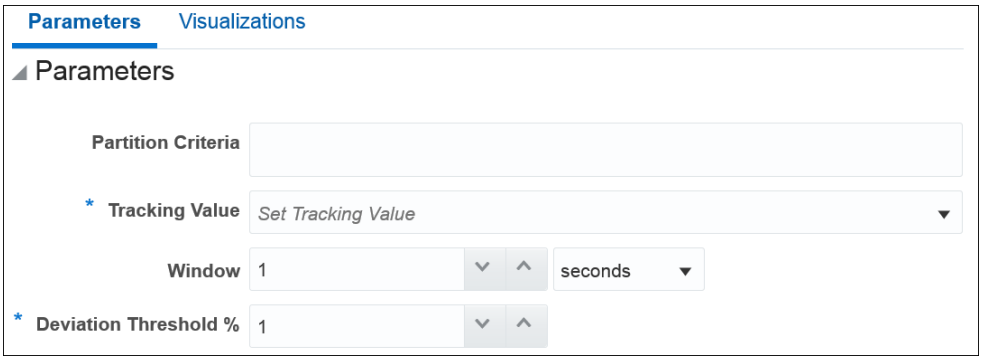

Provide suitable values for the following parameters:

- **Second event stream**: the stream you want to merge with your primary stream. Make sure you select a stream with an identical shape.

The outgoing shape is the same as the incoming shape.

## About the Fluctuation Pattern

Use this pattern to detect when an event data field value changes in a specific upward or downward fashion within a specific time window. For example, use this pattern to identify the variable changes in an Oil Pressure value are maintained within acceptable ranges.




Provide suitable values for the following parameters:

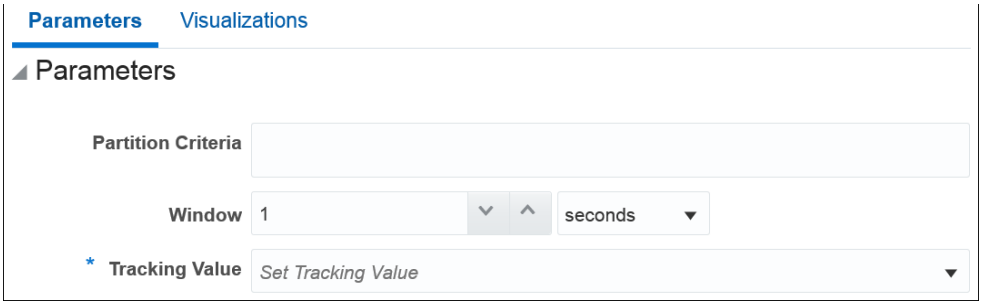
- **Partition Criteria**: the field based on which you want to partition.
- **Tracking Value**: the value is used to track the event data and create a pattern in the live output stream.
- **Window**: a rolling time period, the frequency at which you want to refresh the data.
- **Deviation Threshold %**: value indicates the percentage of deviation you want to be included in the pattern. This is the interval in which the pipeline looks for a matching pattern.

The outgoing shape is same as the incoming shape.

## About the Inverse W Pattern

The Inverse W pattern, also known as a double top chart pattern, is used in the technical analysis of financial trading markets.

Use this pattern when you want to see the financial data in a graphical form.




Provide suitable values for the following parameters:

- **Partition Criteria**: a field to partition your stream by. For example, a ticker symbol.
- **Window**: a time period, within which the values of the designated field are analyzed for the inverse W shape.
- **Tracking value**: a field, whose values are analyzed for the inverse W shape.

### Outgoing Shape

The outgoing shape is based on the incoming shape with an addition of five new fields. The new fields are:

- `firstW`
- `firstPeakW`
- `headInverseW`
- `secondpeakW`
- `lastW`

The new fields correspond to the tracking value terminal points of the inverse W shape discovered in the feed. The original fields correspond to the last event in the inverse W pattern.

## About the Eliminate Duplicates Pattern

The Eliminate Duplicates pattern looks for duplicate events in your stream within a specified time window and removes all but the first occurrence. A duplicate event is an event that has one or more field values identical to values of the same field(s) in another event. It is up to you to specify what fields are analyzed for duplicate values. You can configure the pattern to compare just one field or the whole event.

Use it to get rid of noise in your stream. If you know that your stream contains duplicates that might offset your aggregates, such as counts, use the Eliminate Duplicates pattern to cleanse your data.


Provide suitable values for the following parameters:

- **Duplicate Criteria**: a list of fields, whose values will be compared between events to look for identical values. If all the configured fields have identical values, the second, third, and subsequent events will be dropped.
- **Window**: a time period, within which the duplicates will be discarded. For example, if you set the window to 10 seconds, a duplicate event that arrives 9 seconds after the first one will be discarded, while a duplicate event that arrives 11 seconds after the first one will be accepted and let through.

The outgoing shape is the same as the incoming shape.

## About the Detect Missing Heartbeat Pattern

The Detect Missing Heartbeat pattern discovers simple situations when an expected event is missing.

Use this pattern if you need to detect missing events in your feed. For example, you have a feed when multiple sensors send their readings every 5 seconds. Use this pattern to detect sensors that have stopped sending their readings, which may indicate that the sensor is broken or there is no connection to the sensor.


Provide suitable values for the following parameters:

- **Partition Criteria**:  a field to partition your stream by. For example, your stream contains events issues by a number of sensors. All sensors send the same but individual data. You would want to compare readings of a sensor to previous readings of the same sensor and not just a previous event in your stream, which is very likely to be from a different sensor. Select a field that would uniquely identify your sensors, such as sensor id. This field is optional. For example, if your stream contains readings from just one sensor, you do not need to partition your data.
- **Heartbeat interval**: a time period, within which missing events are detected. If there is no event from a sensor within the heartbeat interval after the last event, an alert is triggered.

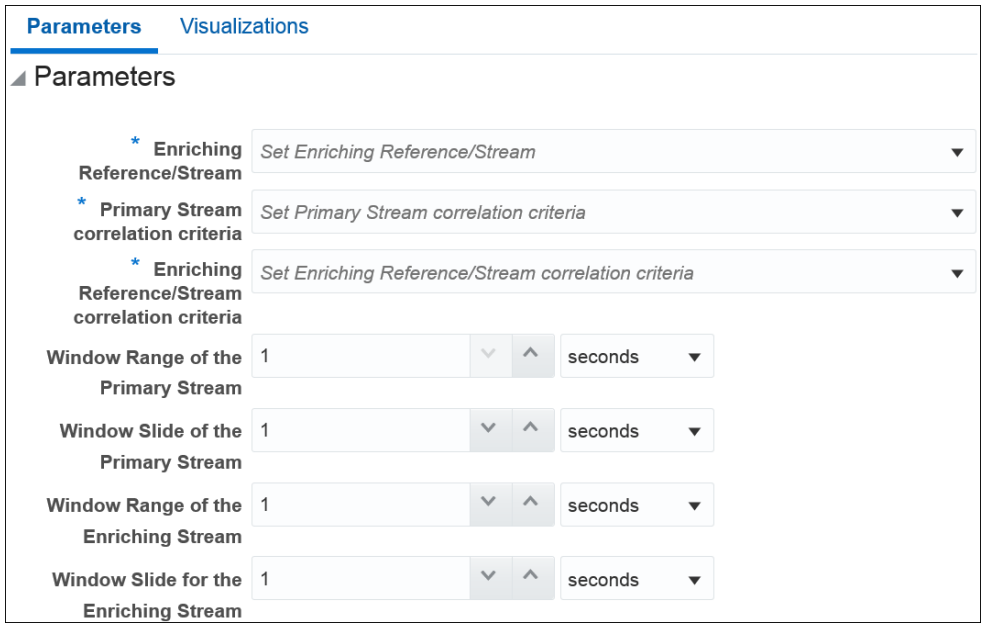
### Outgoing Shape

The outgoing shape is the same as incoming shape. If there are no missing heartbeats, no events are output. If there is a missing heartbeat, the previous event, which was used to calculate the heartbeat interval is output.

## About the Left Outer Join Pattern

The Left Outer join pattern joins your flow with another stream or a reference using the left outer join semantics.

Use this pattern to join a stream or a reference using the left outer join semantics. The result of this pattern always contains the data of the left table even if the join-condition does not find any matching data in the right table.




Provide suitable values for the following parameters:

- **Enriching Reference/Stream:** the stream or reference you want to join to your flow.
- **Correlation Criteria**: fields on which the stream / reference will be joined.
- **Window Range of the Primary Stream**: a rolling time window used to make a collection of events in your primary flow to be joined with the enriching stream / reference.
- **Window Slide of the Primary Stream**: how often the data will be pushed downstream and to the UI.
- **Window Range of the Enriching Stream**: a rolling time window used to make a collection of events in your enriching stream to be joined with the primary flow. Disabled, if a reference is used.
- **Window Slide for the Enriching Stream**: how often the data will be pushed downstream and to the UI. Disabled, if a reference is used.

The outgoing shape is a sum of two incoming shapes.

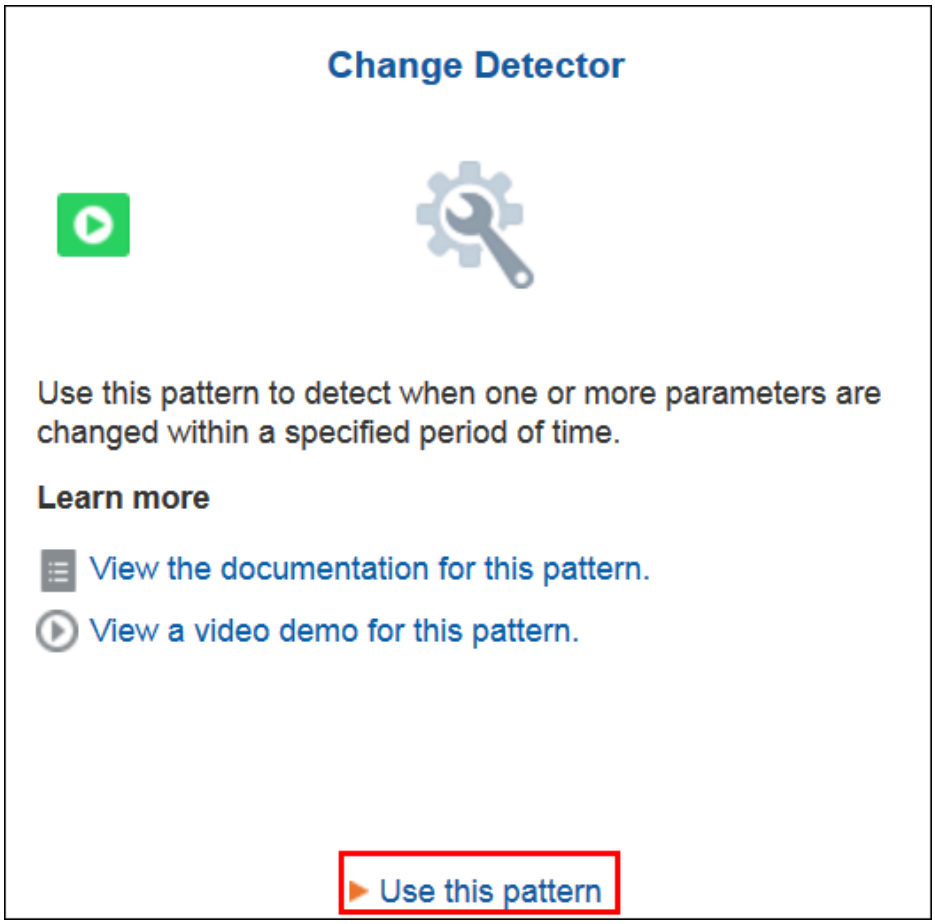
# Create a Pipeline for a Pattern

Instead of creating a pattern stage from within a pipeline, you can also create a pipeline for a pattern directly.

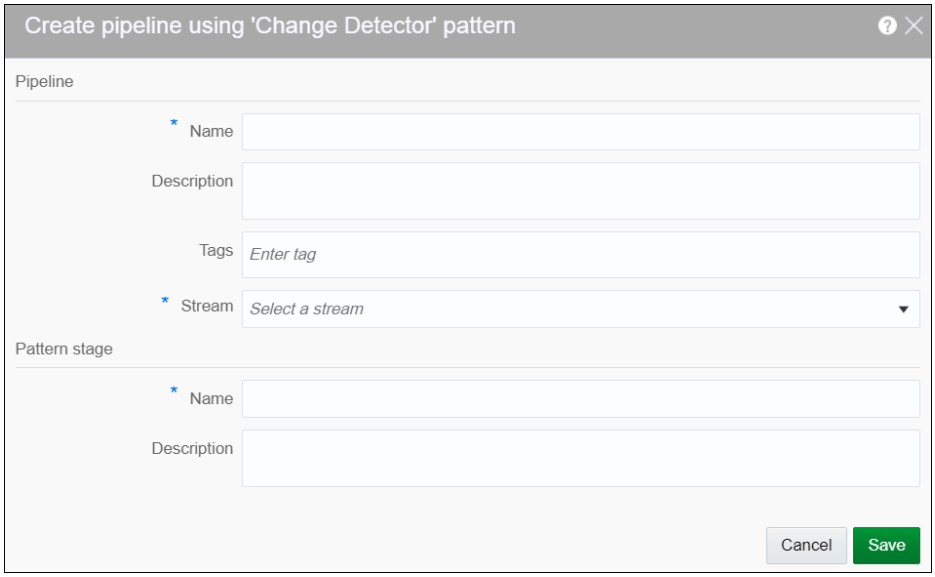
To create a pipeline for a pattern:

1. Click **Patterns** in the left tree on the Home page.

   The Patterns page appears.
2. Scroll through the list of available patterns and select the required pattern.
3. Click **Use this pattern** within the selected pattern tile.




The Create pipeline using <Pattern> dialog box appears.




4. Fill in the details for the metadata in the **pipeline** section.
5. Enter details for the **Pattern Stage**.
6. Click **Save**.

The pipeline editor opens where you can specify the parameters required for the pattern. The pipeline also appears in the Catalog.

5

# Expression Builder Functions

Expression Builder is an editor that allows you to build expressions using various existing functions. The expressions help you in achieving the required results for your pipelines.

**Topics:**



## Bessel Functions

The mathematical cylinder functions for integers are known as Bessel functions.

The following Bessel functions are supported in this release:

| Function Name | Description |
|---|---|
| `BesselI0(x)` | Returns the modified Bessel function of order 0 of the double argument as a double |
| `BesselI0_exp(x)` | Returns the exponentially scaled modified Bessel function of order 0 of the double argument as a double |
| `BesselI1(x)` | Returns the modified Bessel function of order 1 of the double argument as a double |
| `BesselI1_exp(x)` | Returns the exponentially scaled modified Bessel function of order 1 of the double argument as a double |
| `BesselJ(x,x)` | Returns the Bessel function of the first kind of order n of the argument as a double |
| `BesselK(x,x)` | Returns the modified Bessel function of the third kind of order n of the argument as a double |
| `BesselK0_exp(x)` | Returns the exponentially scaled modified Bessel function of the third kind of order 0 of the double argument as a double |
| `BesselK1_exp(x)` | Returns the exponentially scaled modified Bessel function of the third kind of order 1 of the double argument as a double |

| Function Name | Description |
|---|---|
| `BesselY(x)` | Returns the Bessel function of the second kind of order n of the double argument as a double |

# Conversion Functions

The conversion functions help in converting values from one data type to other.

The following conversion functions are supported in this release:

| Function Name | Description |
|---|---|
| `bigdecimal(value1)` | Converts the given value to bigdecimal |
| `boolean(value1)` | Converts the given value to logical |
| `date(value1,value2)` | Converts the given value to datetime |
| `double(value1)` | Converts the given value to double |
| `float(value1)` | Converts the given value to float |
| `int(value1)` | Converts the given value to integer |
| `long(value1)` | Converts the given value to long |
| `string(value1,value2)` | Converts the given value to string |

## boolean(value1)

Converts the input argument value to logical. The input argument can be one of the following data type: big integer or integer. Returned value type will be Boolean.

**Examples**

| Function | Result |
|---|---|
| `boolean(5)` | TRUE |
| `boolean(0)` | FALSE |
| `boolean(NULL)` | TRUE |
| `boolean()` | TRUE |
| `boolean(-5)` | TRUE |

## double(value1)

Converts the input argument value to double. The input argument can be one of the following data types: integer, big integer, double, text or float. Returned value will be a double-precision floating-point number.

**Examples**

| Function | Result |
|---|---|
| `double("3.14")` | 3.14E+0 |
| `double(1234.56)` | 1.235E+003 |

## float(value1)

Converts the input argument value to float. The input argument can be one of the following data types: integer, big integer, double, text or float. Returned value will be a single-precision floating-point number.

**Examples**

| Function | Result |
|---|---|
| `float("3.14")` | 3.14E+0 |
| `float(1234.56)` | 1.235E+003 |

# Date Functions

The following date functions are supported in this release:

| Function Name | Description |
|---|---|
| `day(date)` | Returns day of the date |
| `eventtimestamp()` | Returns event timestamp from stream |
| `hour(date)` | Returns hour of the date |
| `minute(date)` | Returns minute of the date |
| `month(date)` | Returns month of the date |
| `nanosecond(date)` | Returns nanosecond of the date |
| `second(date)` | Returns second of the date |
| `systimestamp()` | Returns the system's timestamp on which the application is running |
| `timeformat(value1,value2)` | Returns the provided timestamp in required time format |
| `timestamp()` | Returns the current output time |
| `year(date)` | Returns year of the date |

## Day(date)

`day(date)` function takes as an argument any one of the following data types: time interval or timestamp. The returned value represents the day of the week in the time represented by this date object. Returns a big integer indicating the day of the week represented by this date.

**Examples**

If Sunday=0, Monday=1 and so on, then:

| Function | Result |
|---|---|
| `day(12/06/17 09:15:22 AM)` | 3 |
| `day(2017:11:23 11:20:25 PM)` | 4 |

## hour(date)

`hour(date)` function takes as an argument any one of the following data types: time interval or timestamp. The returned value represents the hour in the time represented by this date object. Returns a big integer indicating the hour of the time represented by this date.

**Examples**

| Function | Result |
|---|---|
| `hour(12/06/17 09:15:22 AM)` | 09 |
| `hour(2015:07:21 12:45:35 PM)` | 12 |

## minute(date)

`minute(date)` function takes as an argument any one of the following data types: time interval or timestamp. The returned value represents the minutes in the time represented by this date object. Returns a big integer indicating the minutes of the time represented by this date.

**Examples**

| Function | Result |
|---|---|
| `minute(12/06/17 09:15:22 AM)` | 15 |
| `minute(2015:07:21 12:45:35 PM)` | 45 |

## month(date)

`month(date)` function takes as an argument any one of the following data types: time interval or timestamp. The returned value represents the month of the year that contains or begins with the instant in time represented by this date object. Returns a big integer indicating the month of the year represented by this date.

**Examples**

| Function | Result |
|---|---|
| `month(12/06/17 09:15:22 AM)` | 12 |
| `month(2017:09:23 11:20:25 AM)` | 9 |

## second(date)

`second(date)` function takes as an argument any one of the following data types: time interval or timestamp. The returned value represents the seconds of the instant in time represented by this date object. Returns a big integer indicating the seconds of the time represented by this date.

**Example**

| Function | Result |
|---|---|
| `second(12/06/17 09:15:22 AM)` | 22 |
| `second((2015:07:21 12:45:35 PM)` | 35 |

## Year(date)

`year(date)` function takes as an argument any one of the following data types: time interval or time stamp. The returned value represents the year of the instant in time represented by this date object. Returns a big integer indicating the year represented by this date.

**Examples**

| Function | Result |
|---|---|
| `year(12/06/17 09:15:22 AM)` | 17 |
| `year(2015:07:21 12:45:35 PM)` | 2015 |

# Geometry Functions

The Geometry functions allow you to convert the given values into a geometrical shape.

The following interval functions are supported in this release:

| Function Name | Description |
|---|---|
| `CreatePoint(lat,long,SRID)` | Returns a 2–dimensional point type geometry from the given latitude and longitude. The default SRID is 8307.<br>The return value is of the datatype `sdo geometry`. |
| `distance(lat1,long1,lat2,long2,SRID)` | Returns distance between the first set of latitude, longitude and the second set of latitude, longitude values. The default SRID is 8307.<br>The return value is of the datatype `double`. |

# Interval Functions

The Interval functions help you in calculating time interval from given values.

The following interval functions are supported in this release:

| Function Name | Description |
|---|---|
| `numtodsinterval(n,interval_unit)` | Converts the given value to an `INTERVAL DAY TO SECOND` literal. The value of the `interval_unit` specifies the unit of `n` and must resolve to one of the string values: *DAY*, *HOUR*, *MINUTE*, or *SECOND*.<br>The return value is of the datatype `interval`. |
| `to_dsinterval(string)` | Converts a string in format `DD HH:MM:SS` into a `INTERVAL DAY TO SECOND` data type. The `DD` indicates the number of days between 0 to 99. The `HH:MM:SS` indicates the number of hours, minutes and seconds in the interval from 0:0:0 to 23:59:59.999999. The seconds part can accept upto six decimal places.<br>The return value is of the datatype `interval`. |

# Math Functions

The math functions allow you to perform various mathematical operations and calculations ranging from simple to complex.

The following math functions are supported in this release:

| Function Name | Description |
|---|---|
| `IEEEremainder(value1,value2)` | Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard |
| `abs(value1)` | Returns the absolute value of a number |
| `acos(value1)` | Returns arc cosine of a value |
| `asin(value1)` | Returns arc sine of a value |
| `atan(value1)` | Returns arc tangent of a value |
| `atan2(arg1,arg2)` | Returns polar angle of a point (`arg2`, `arg1`) |
| `binomial(base,power)` | Returns binomial coefficient of the base raised to the specified power |
| `bitMaskWithBitsSetFromTo(x)` | BitMask with BitsSet (From, To) |
| `cbrt(value1)` | Returns cubic root of the specified value |
| `ceil(value1)` | Rounds to ceiling |
| `copySign(value1,value2)` | Returns the first floating-point argument with the sign of the second floating-point argument |

| Function Name | Description |
|---|---|
| `cos(value1)` | Returns cosine of a value |
| `cosh(value1)` | Returns cosine hyperbolic of a value |
| `exp(x)` | Returns exponent of a value |
| `expm1(x)` | More precise equivalent of `exp(x)`; Returns 1 when x is around zero |
| `factorial(value1)` | Returns factorial of a natural number |
| `floor(value1)` | Rounds to floor |
| `getExponent(value1)` | Returns the unbiased exponent used in the representation of a double |
| `getSeedAtRowColumn(value1,value2)` | Returns a deterministic seed as an integer from a (seemingly gigantic) matrix of predefined seeds |
| `hash(value1)` | Returns an integer hashcode for the specified double value |
| `hypot(value1,value2)` | Returns square root of sum of squares of the two arguments |
| `leastSignificantBit(value1)` | Returns the least significant 64 bits of this UUID's 128 bit value |
| `log(value1,value2)` | Calculates the log value of the given argument to the given base |
| `log1(value1)` | Returns the natural logarithm of a number |
| `log10(value1)` | Calculates the log value of the given argument to base 10 |
| `log2(value1)` | Calculates the log value of the given argument to base 2 |
| `logFactorial(value1)` | Returns the natural logarithm (base e) of the factorial of its integer argument as a double |
| `longFactorial(value1)` | Returns the factorial of its integer argument (in the range k >= 0 && k < 21) as a long |
| `maximum(value1,value2)` | Returns the maximum of 2 arguments |
| `minimum(value1,value2)` | Returns the minimum of 2 arguments |
| `mod(value1,value2)` | Returns modulo of a number |
| `mosttSignificantBit(value1)` | Returns the most significant 64 bits of this UUID's 128 bit value |
| `nextAfter(value1,value2)` | Returns the floating-point number adjacent to the first argument in the direction of the second argument |
| `nextDown(value1)` | Returns the floating-point value adjacent to the input argument in the direction of negative infinity |
| `nextUp(value1)` | Returns the floating-point value adjacent to the input argument in the direction of positive infinity |
| `Pow(m,n)` | Returns m raised to the nth power |
| `rint(value1)` | Returns the double value that is closest in value to the argument and is equal to a mathematical integer |
| `round(value1)` | Rounds to the nearest integral value |

| Function Name | Description |
|---|---|
| `Scalb(d,scaleFactor)` | Returns d × 2scaleFactor rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set |
| `signum(value1)` | Returns signum of an argument as a double value |
| `sin(value1)` | Returns sine of a value |
| `sinh(value1)` | Returns sine hyperbolic of a value |
| `sqrt(value1)` | Returns square root of a value |
| `stirlingCorrection(value1)` | Returns the correction term of the Stirling approximation of the natural logarithm (base e) of the factorial of the integer argument as a double |
| `tan(value1)` | Returns tangent of a value |
| `tanh(value1)` | Returns tangent hyperbolic of a value |
| `toDegrees(value1)` | Converts the argument value to degrees |
| `toRadians(value1)` | Returns the measurement of the angle in radians |
| `ulp(value1)` | Returns the size of an ulp of the argument |

## maximum(value1, value2)

Returns the maximum of two arguments. The first argument is a value to compare with the second argument's value and can be any one of the following data type: big integer, double, interval, integer, float. The second argument is a value to compare with the first argument's value and can be any one of the following data type: big integer, double, interval, integer, float.

**Examples**

| Function | Result |
|---|---|
| `maximum(1999220,1997220)` | 1999220 |
| `maximum(135.45, 135.50)` | 135.50 |

> **Note:**
>
> If the user provides two different data types as input arguments, then Stream Analytics does implicit conversion to convert one of the argument to the other argument's type.

## minimum(value1, value2)

Returns the minimum of two arguments. The first argument is a value to compare with the second argument's value and can be any one of the following data type: big integer, double, interval, integer, float. The second argument is a value to compare with the first argument's value and can be any one of the following data type: big integer, double, interval, integer, float.

**Examples**

| Function | Result |
|---|---|
| `minimum(16324, 16321)` | 16321 |
| `minimum(3.16, 3.10)` | 3.10 |

> **Note:**
>
> If the user provides two different data types as arguments, then Stream Analytics does implicit conversion to convert one argument to the other argument's type.

## round(value1)

Rounds the argument value to the nearest integer value. The input argument can be of the following data types: big integer, double, integer, float.

**Examples**

| Function | Result |
|---|---|
| `round(7.16)` | 7 |
| `round(38.941)` | 39 |
| `round(3.5)` | 4 |

## toDegrees(value1)

Converts the argument value to degrees. The input argument is an angle in radians and can be of type double. The returned value will be the measurement of the angle in degrees and is of type double.

**Examples**

| Function | Result |
|---|---|
| `toDegrees(3.14)` | 180.0 |
| `toDegrees(0.785)` | 45.0 |

## toRadians(value1)

Converts the argument value to radians. The input argument is an angle in degrees and can be of type double. The returned value will be the measurement of the angle in radians and is of type double.

**Examples**

| Function | Result |
|---|---|
| `toRadians(180.0)` | 3.14 |
| `toRadians(45.0)` | 0.785 |

# Null-related Functions

The following null-related functions are supported in this release:

| Function Name | Description |
|---|---|
| `nvl(value1,value2)` | Replaces null with a value of the same type |

## nvl(value1, value2)

`nvl` lets you replace null (returned as a blank) with a value of the same type as the first argument. For example, in a list of employees and commission, you can substitute *Not Applicable* if the employee receives no commission using the `nvl(value1,value2)` function as `nvl(Not Applicable,Commission)`.

**Example**

| Function | Result |
|---|---|
| `nvl(Not Applicable,Commission)` | Not Applicable |

# Statistical Functions

Statistical functions help you in calculating the statistics of different values.

The following statistical functions are supported in this release:

| Function Name | Description |
|---|---|
| `beta1(value1,value2,value3)` | Returns the area from zero to `value3` under the beta density function |
| `betaComplemented(value1,value2,value3)` | Returns the area under the right hand tail (from `value3` to infinity) of the beta density function |
| `binomial2(value1,value2,value3)` | Returns the sum of the terms 0 through `value1` of the Binomial probability density. All arguments must be positive. |
| `binomialComplemented(value1,value2,value3)` | Returns the sum of the terms `value1+1` through `value2` of the binomial probability density. All arguments must be positive. |
| `chiSquare(value1,value2)` | Returns the area under the left hand tail (from 0 to `value2`) of the chi square probability density function with `value1` degrees of freedom. The arguments must both be positive. |

| Function Name | Description |
|---|---|
| `chiSquareComplemented(value1,value2)` | Returns the area under the right hand tail (from `value2` to infinity) of the chi square probability density function with `value1` degrees of freedom. The arguments must both be positive. |
| `errorFunction(value1)` | Returns the error function of the normal distribution |
| `errorFunctionComplemented(value1)` | Returns the complementary error function of the normal distribution |
| `gamma(value1,value2,value3)` | Returns the gamma function of the arguments |
| `gammaComplemented(value1,value2,value3)` | Returns the integral from `value3` to infinity of the gamma probability density function |
| `incompleteBeta(value1,value2,value3)` | Returns the incomplete beta function evaluated from zero to `value3` |
| `incompleteGamma(value1,value2)` | Returns the incomplete gamma function |
| `incompleteGammaComplement(value1,value2)` | Returns the complemented incomplete gamma function |
| `logGamma(value1)` | Returns the natural logarithm of the gamma function |
| `negativeBinomial(value1,value2,value3)` | Returns the sum of the terms 0 through `value1` of the negative binomial distribution. All arguments must be positive. |
| `negativeBinomialComplemented(value1,value2,value3)` | Returns the sum of the terms `value1+1` to infinity of the negative binomial distribution. All arguments must be positive. |
| `normal(value1,value2,value3)` | Returns the area under the normal (Gaussian) probability density function, integrated from minus infinity to `value1` (assumes mean is zero, variance is one) |
| `normalInverse(value1)` | Returns the value for which the area under the normal (Gaussian) probability density function is equal to the argument `value1` (assumes mean is zero, variance is one) |
| `poisson(value1,value2)` | Returns the sum of the first `value1` terms of the Poisson distribution. The arguments must both be positive. |
| `poissonComplemented(value1,value2)` | Returns the sum of the terms `value1+1` to infinity of the poisson distribution |
| `studentT(value1,value2)` | Returns the integral from minus infinity to `value2` of the Student-t distribution with `value1` > 0 degrees of freedom |
| `studentTInverse(value1,value2)` | Returns the value, for which the area under the Student-t probability density function is equal to `1-value1/2`. The function uses the studentT function to determine the return value iteratively. |

# String Functions

The following String functions are supported in this release:

| Function Name | Description |
|---|---|
| coalesce(value1,...) | Returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null |
| concat(value1,...) | Returns concatenation of values converted to strings |
| indexof(string,match) | Returns first index of \'match\' in \'string\'or 1 if not found |
| initcap(value1) | Returns a specified text expression, with the first letter of each word in uppercase and all other letters in lowercase |
| length(value1) | Returns the length of the specified string |
| like(value1,value2) | Returns a matching pattern |
| lower(value1) | Converts the given string to lower case |
| lpad(value1,value2,value3) | Pads the left side of a string with a specific set of characters (when string1 is not null) |
| ltrim(value1,value2) | Removes all specified characters from the left hand side of a string |
| replace(string,match,replacement) | Replaces all \'match\' with \'replacement\' in \'string\' |
| rpad(value1,value2,value3) | Pads the right side of a string with a specific set of characters (when string1 is not null) |
| rtrim(value1,value2) | Removes all specified characters from the right hand side of a string |
| substr(string,from) | Returns substring of a 'string' when indices are between 'from' (inclusive) and up to the end of the string |
| substring(string,from,to) | Returns substring of a \'string\' when indices are between \'from\' (inclusive) and \'to\' (exclusive) |
| translate(value1,value2,value3) | Replaces a sequence of characters in a string with another set of characters. However, it replaces a single character at a time. |
| upper(value1) | Converts given string to uppercase |

# coalesce(value1,... )

coalesce returns the first non-null expression in the list of expressions. You must specify at least two expressions. If all expressions evaluate to null then the coalesce function will return null.

For example:

In coalesce(expr1,expr2):

- If expr1 is not null then the function returns expr1.
- If expr1 is null then the function returns expr2.
- If expr1 and expr2 are null then the function returns null.

In coalesce(expr1,expr2,.....,exprn)

- If expr1 is not null then the function returns expr1.

- If `expr1` is null then the function returns `expr2`.
- If `expr1` and `expr2` are null then the function returns the next non-null expression.

## length(value1)

Returns the length in characters of the string passed as an input argument. The input argument is of the data type text. The returned value is an integer representing the total length of the string.

If `value1` is null, then `length(value1)` returns null.

If `value1` is an empty string, then `length(value1)` returns null.

**Examples**

| Function | Result |
|---|---|
| `length(“one”)` | 3 |
| `length()` | ERROR: Function has invalid parameters. |
| `length(“john”)` | 4 |
| `length(” “)` | NULL |
| `length(null)` | NULL |
| `length(“firstname.lastname@example.com”)` | 30 |

## lower(value1)

Converts a string to all lower-case characters. The input argument is of the data type text. The returned value is the lowercase of the specified string.

**Examples**

| Function | Result |
|---|---|
| `lower(“PRODUCT”)` | product |
| `lower(“ABCdef”)` | abcdef |
| `lower(“abc”)` | abc |

## replace(string, match, replacement)

Replaces all `match` characters in a string with `replacement` characters. The first input argument is the `string` and is of the data type text. The second argument is the `match` and is of the data type text. The third argument is `replacement` and is of data type text. The returned value is a text in which the third string argument (`replacement`) replaces the second string argument (`match`).

If `match` is not found in the string, then the original string will be returned.

**Examples**

| Function | Result |
|---|---|
| `replace(“aabbccdd”,”cc”,”ff”)` | aabbffdd |
| `replace(“aabbcccdd”,”cc”,”ff”)` | aabbffcdd |
| `replace(“aabbddee”,”cc”,”ff”)` | aabbddee |

## substring(string, from, to)

Returns a substring of a string when indices are between `from` (inclusive) and `to` (exclusive). The first input argument is the `string` and is of the data type text. The second argument is the start index and is an integer. The third argument is the finish index and is an integer. The returned value is a substring and is of type text.

**Examples**

| Function | Result |
|---|---|
| `substring(“abcdefgh”,3,7)` | cdef |
| `substring(“abcdefgh”,1,6)` | abcde |

## upper(value1)

Converts a string to all upper-case characters. The input argument is of the data type text. The returned value is the uppercase of the specified string.

**Examples**

| Function | Result |
|---|---|
| `upper(“name”)` | NAME |
| `upper(“abcdEFGH”)` | ABCDEFGH |
| `upper(“ABCD”)` | ABCD |

# 6
# Troubleshoot Stream Analytics

After you provision and run the pipeline, sometimes you may encounter issues with the pipeline. Some of those issues are explained here. Each pipeline is composed of various stages. A stage can be a stream, query, or pattern.

**Topics:**



## Troubleshoot Live Output

For every pipeline, there will be one Spark streaming pipeline running on Spark Cluster. If a Stream Analytics pipeline uses one or more Query Stage or Pattern Stage, then the pipeline will run one or more continuous query for each of these stages.

For more information about continuous query, see Understanding Oracle CQL.

If there are no output events in Live Output Table for Query Stage or Pattern Stage, use the following steps to determine or narrow down the problem:

1. Ensure that Pipeline is Deployed Successfully
2. Ensure that the Input Stream is Supplying Continuous Stream of Events to the Pipeline
3. Ensure that CQL Queries for Each Query Stage Emit Output
4. Ensure that the Output of Stage is Available

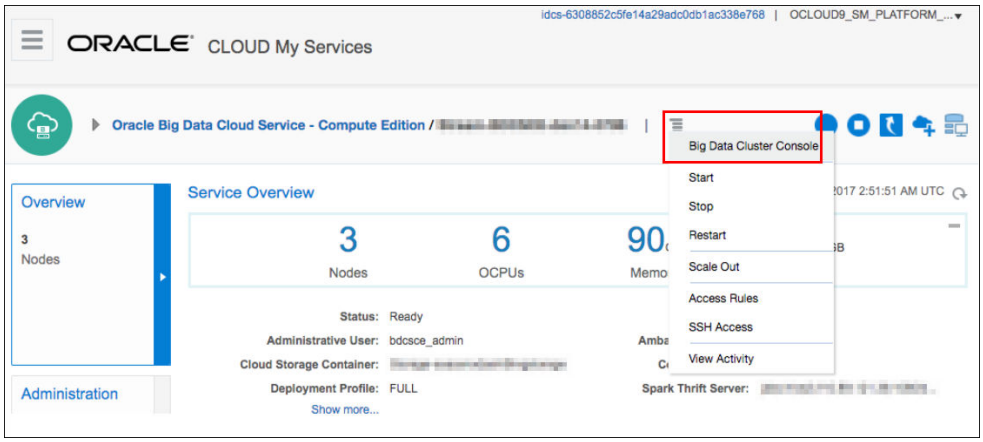
### Ensure that Pipeline is Deployed Successfully

You can deploy pipelines to any Spark Cluster (version 1.6).

Follow the steps in the below sections to verify that the pipeline is deployed and running successfully on Spark cluster.

**Verify pipeline Deployment on Oracle Big Data Cloud Service - Compute Edition based Spark Cluster**

1. Go to PSM user interface and open the home page for Oracle Big Data Cloud Service (BDCSCE) instance.
2. Click on the hamburger menu next to instance name and then click **Big Data Cluster Console**.




3. Enter the login credentials and open the Big Data Cluster Console home page.
4. Navigate to **Jobs** tab.

   You can see a list of jobs. Each job corresponds to a spark pipeline running on your BDCSCE cluster.

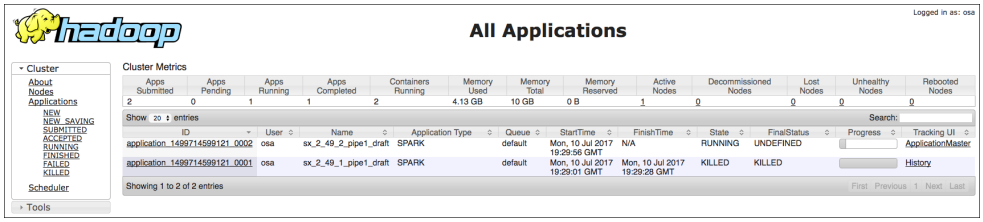

5. Find the entry corresponding to your pipeline and check the status. For more information, see Determine the Spark Application Name Corresponding to a Pipeline.

   If you see the status as `Running`, then the pipeline is currently deployed and running successfully.
6. Click the hamburger menu corresponding to the required job to fetch logs and click Logs to get container wise logs.

   You can download these files for further debugging.

### Verify pipeline Deployment on Apache Spark Installation based Spark Cluster

1. Open Spark Master user interface.




2. Find the entry corresponding to your pipeline and check the status. For more information, see Determine the Spark Application Name Corresponding to a Pipeline.

   If you see the status as `Running`, then the pipeline is currently deployed and running successfully.

# Ensure that the Input Stream is Supplying Continuous Stream of Events to the Pipeline

You must have a continuous supply of events from the input stream.

1. Go to the **Catalog**.
2. Locate and click the stream you want to troubleshoot.
3. Check the value of the **topicName** property under the **Source Type Parameters** section.
4. Listen to the Kafka topic where the input stream for the pipeline is received.

   Since this topic is created using Kafka APIs, you cannot consume this topic with REST APIs.

   a. Listen to the Kafka topic hosted on Oracle Event Hub Cloud Service. You must use Apache Kafka utilities or any other relevant tool to listed to the topic.

      Follow these steps to listen to Kafka topic:

      i. Determine the Zookeeper Address. — Go to Oracle Event Hub Cloud Service Platform home page. Find the IP Address of Zookeeper.

      ii. Use following command to listen the Kafka topic:

      ```
      ./kafka-console-consumer.sh --zookeeper IPAddress:2181 --topic
      nano
      ```

   b. Listen to the Kafka topic hosted on a standard Apache Kafka installation.

      You can listen to the Kafka topic using utilities from a Kafka Installation. `kafka-console-consumer.sh` is a utility script available as part of any Kafka installation.

      Follow these steps to listen to Kafka topic:

      i. Determine the Zookeeper Address from Apache Kafka Installation based Cluster.

      ii. Use the following command to listen the Kafka topic:

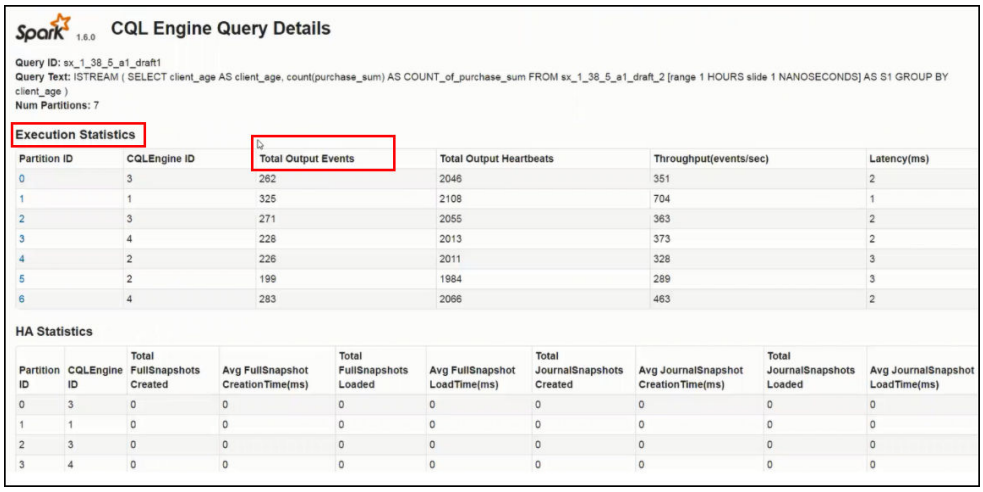
      ```
      ./kafka-console-consumer.sh --zookeeper IPAddress:2181 --topic
      nano
      ```

# Ensure that CQL Queries for Each Query Stage Emit Output

Check if the CQL queries are emitting output events to monitor CQL Queries using CQL Engine Metrics.

Follow these steps to check the output events:

1. Open CQL Engine Query Details page. For more information, see Access CQL Engine Metrics.
2. Check that at least one partition has **Total Output Events** greater than zero under the **Execution Statistics** section.




If your query is running without any error and input data is continuously coming, then the **Total Output Events** will keep rising.

# Ensure that the Output of Stage is Available

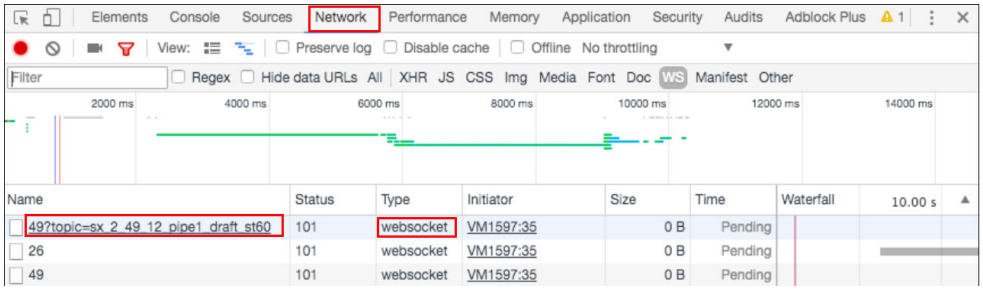
One of the essential things required to troubleshoot a pipeline is to ensure that the output of stage is available in monitor topic.

Follow these steps to check if the output stream is available in the monitor topic:

1. Ensure that you stay in the pipeline Editor and don't click **Done**. Else, the pipeline will be undeployed.
2. Right-click anywhere in the browser and click **Inspect**.
3. Select **Network** from the top tab and then select **WS**.
4. Refresh the browser.

   New websocket connections are created.
5. Locate a websocket whose URL has a parameter with name `topic`.

   The value of the topic param is the name of the Kafka topic where the output of this stage is pushed.




6. Listen to the Kafka topic where output of the stage is being pushed.

   Since this topic is created using Kafka APIs, you cannot consume this topic with REST APIs. Follow these steps to listen to the Kafka topic:

   a. Listen to the Kafka topic hosted on Oracle Event Hub Cloud Service. You must use Apache Kafka utilities or any other relevant tool to listed to the topic.

      Follow these steps to listen to Kafka topic:

i. Determine the Zookeeper Address. — Go to Oracle Event Hub Cloud Service Platform home page. Find the IP Address of Zookeeper.

ii. Use following command to listen the Kafka topic:

```
./kafka-console-consumer.sh --zookeeper IPAddress:2181 --topic
sx_2_49_12_pipe1_draft_st60
```

b. Listen to the Kafka topic hosted on a standard Apache Kafka installation.

You can listen to the Kafka topic using utilities from a Kafka Installation. `kafka-console-consumer.sh` is a utility script available as part of any Kafka installation.

Follow these steps to listen to Kafka topic:

i. Determine the Zookeeper Address from Apache Kafka Installation based Cluster.

ii. Use following command to listen the Kafka topic:

```
./kafka-console-consumer.sh --zookeeper IPAddress:2181 --topic
sx_2_49_12_pipe1_draft_st60
```

**Example 6-1 Example Title**

(Optional) Use sections to add and organize related examples when including an example with a concept topic.

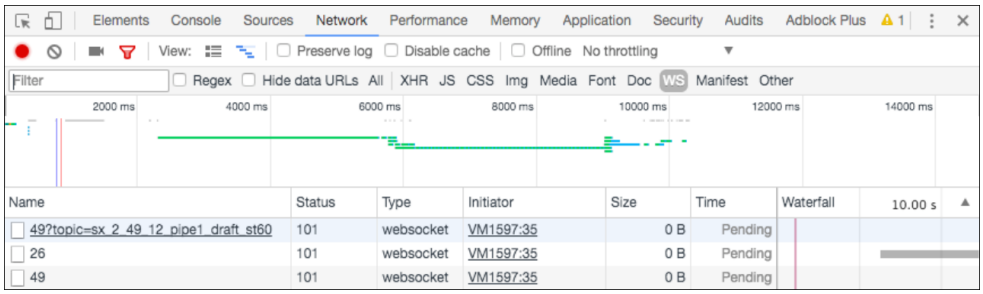
# Determine the Spark Application Name Corresponding to a Pipeline

You can perform the following steps to check if the output stream is available in monitor topic.

1. Navigate to **Catalog**.
2. Open the required pipeline.
3. Ensure that you stay in pipeline editor and do not click **Done**. Otherwise the pipeline gets undeployed.
4. Right-click anywhere in the browser and select **Inspect**.
5. Go to WS tab under the **Network** tab.
6. Refresh the browser.

   New websocket connections are created.
7. Locate a websocket whose URL has a parameter with the name `topic`.

   The value of topic `param` is the name of Kafka Topic where the output of this stage (query or pattern) is pushed.




The topic name is `AppName_StageId`. The pipeline name can be derived from topic name by removing the `_StageID` from topic name. In the above snapshot, the pipeline name is `sx_2_49_12_pipe1_draft`.

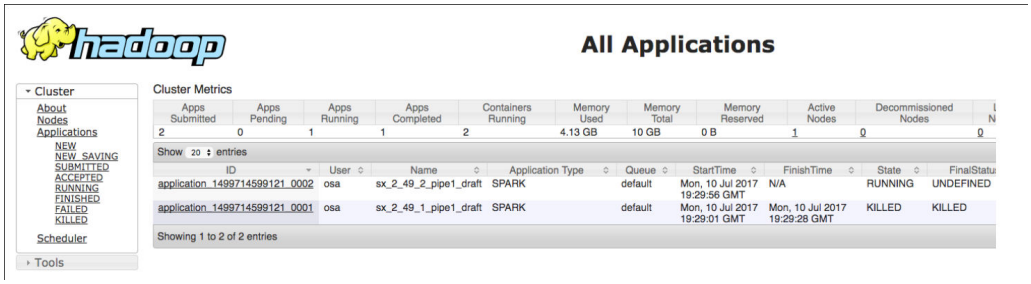
# Access CQL Engine Metrics

When a pipeline with a query or pattern stage is deployed to a Spark cluster, you can perform the complex event processing using a set of CQL Engine Metrics running inside the spark cluster.

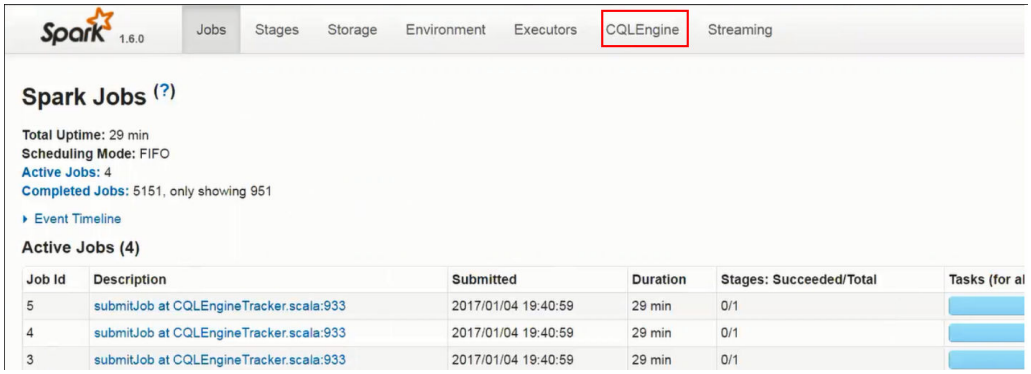
Use CQL queries which can do aggregate, correlate, filter, and pattern matching over a stream of events. Spark provides an out-of-the-box pipeline UI (commonly running on `<host>:4040`) that can help users to monitor a running Spark Streaming pipeline. As CQL queries also run as part of Spark Streaming pipeline, the Spark pipeline UI is extended to include monitoring capabilities of CQL queries.

To access CQL Engine metrics:

1. Create a pipeline with at least one query or pattern stage.
2. Navigate to Spark Master User Interface.

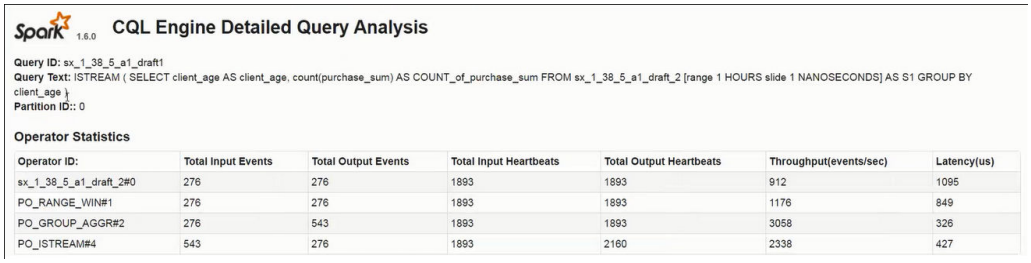



3. Click the **CQL Engine** tab.



You can see the details of all queries running inside a Spark CQL pipeline. This page also shows various streams/relations and external relations registered as part of the pipeline.

4. Click any query to see the details of that query. the query details page shows partition-wise details about a particular running query.
5. Click the specific partition link to determine further details about query plan and operator level details. This page shows the operator level details of a query processing a particular partition.



Spark 1.6.0 **CQL Engine Detailed Query Analysis**

**Query ID:** sx_1_38_5_a1_draft1
**Query Text:** ISTREAM ( SELECT client_age AS client_age, count(purchase_sum) AS COUNT_of_purchase_sum FROM sx_1_38_5_a1_draft_2 [range 1 HOURS slide 1 NANOSECONDS] AS S1 GROUP BY client_age )
**Partition ID::** 0

**Operator Statistics**

| Operator ID: | Total Input Events | Total Output Events | Total Input Heartbeats | Total Output Heartbeats | Throughput(events/sec) | Latency(us) |
|---|---|---|---|---|---|---|
| sx_1_38_5_a1_draft_2#0 | 276 | 276 | 1893 | 1893 | 912 | 1095 |
| PO_RANGE_WIN#1 | 276 | 276 | 1893 | 1893 | 1176 | 849 |
| PO_GROUP_AGGR#2 | 276 | 543 | 1893 | 1893 | 3058 | 326 |
| PO_ISTREAM#4 | 543 | 276 | 1893 | 2160 | 2338 | 427 |

# Troubleshoot Pipeline Deployment

Sometimes pipeline deployment fails with the following exception:

```
Spark pipeline did not start successfully after 60000 ms.
```

This exception usually occurs when you do not have free resources on your cluster.

**Workaround:**

Use external Spark cluster or get better machine and configure the cluster with more resources.